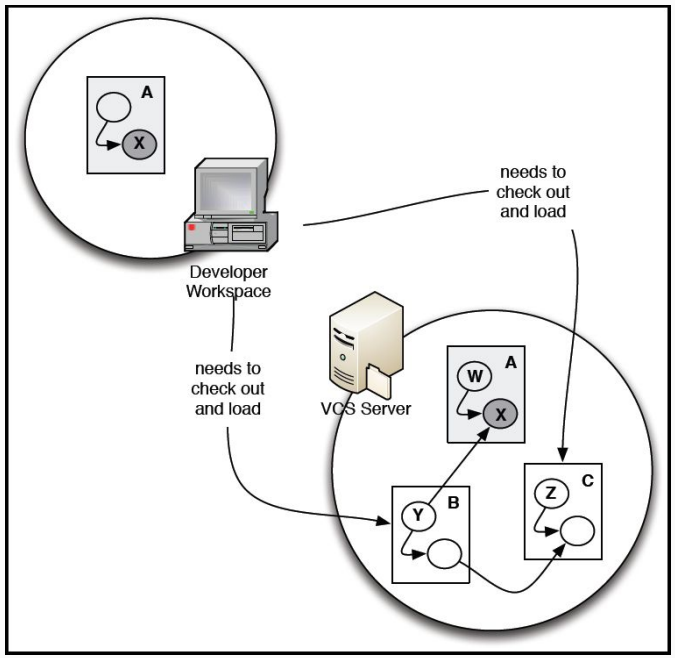
Taming Large Models with Hawk and NeoEMF

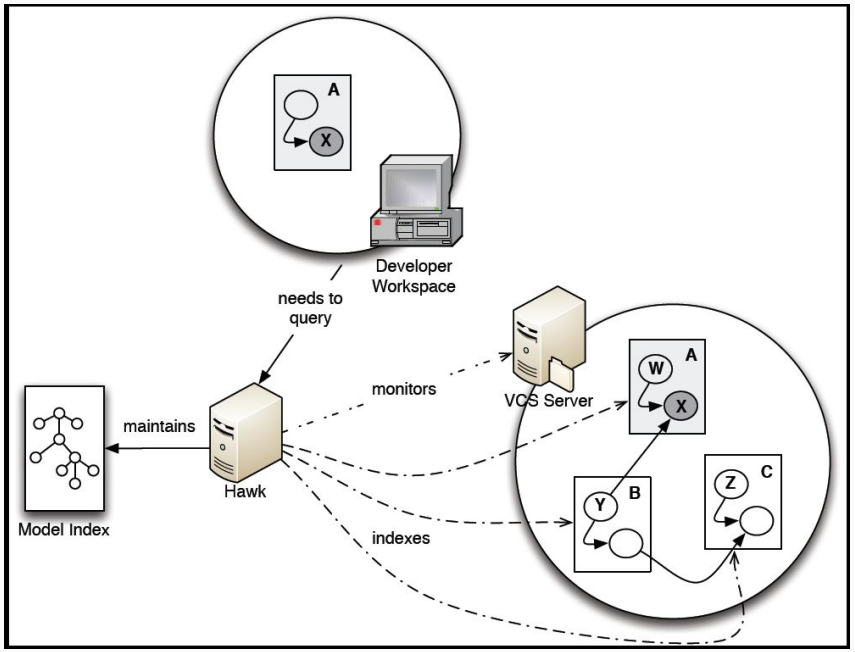
A. García-Domínguez, D. S. Kolovos, K. Barmpis, G. Daniel, G. Sunyé

MoDELS'2018, 14–19 October 2018

Hawk

Hawk: indexing for fast querying over fragment collections





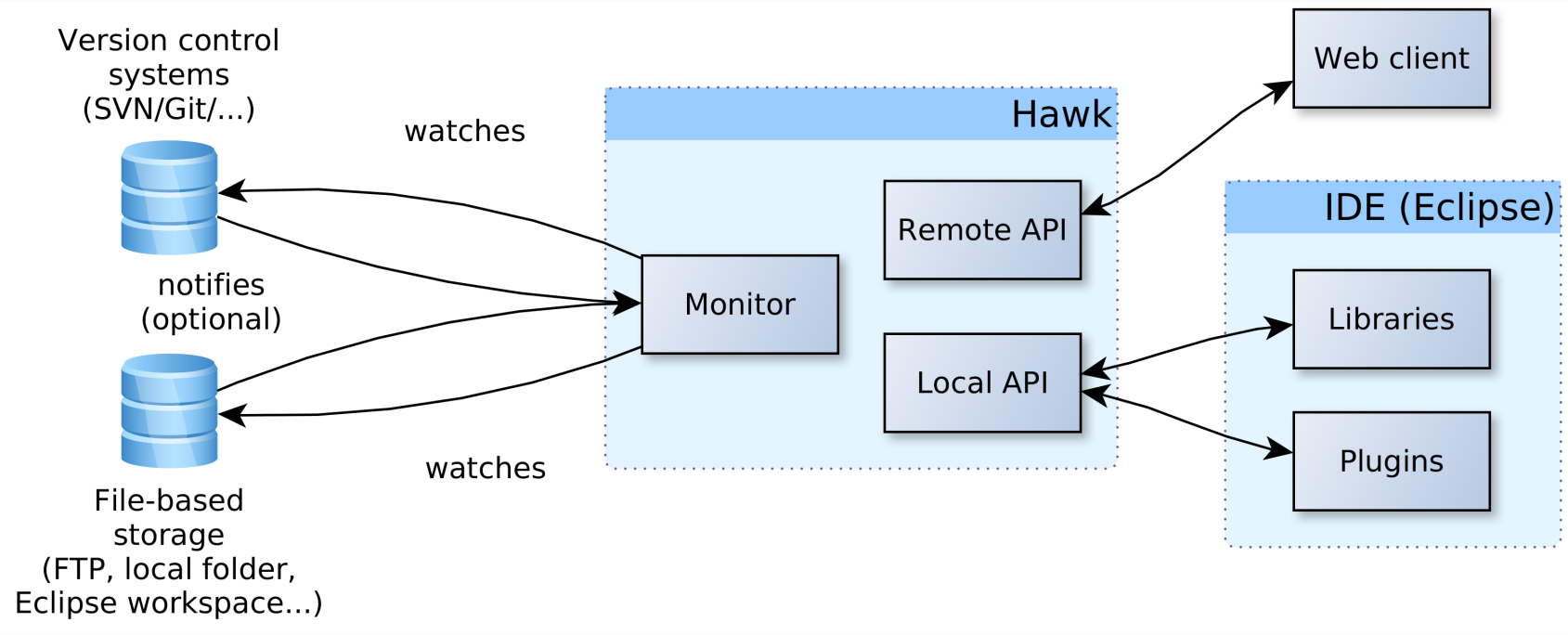
Usual approach

1. Check out **all** files from VCS
2. Load fragments into memory
3. Run query (might go over all fragments)

With Hawk

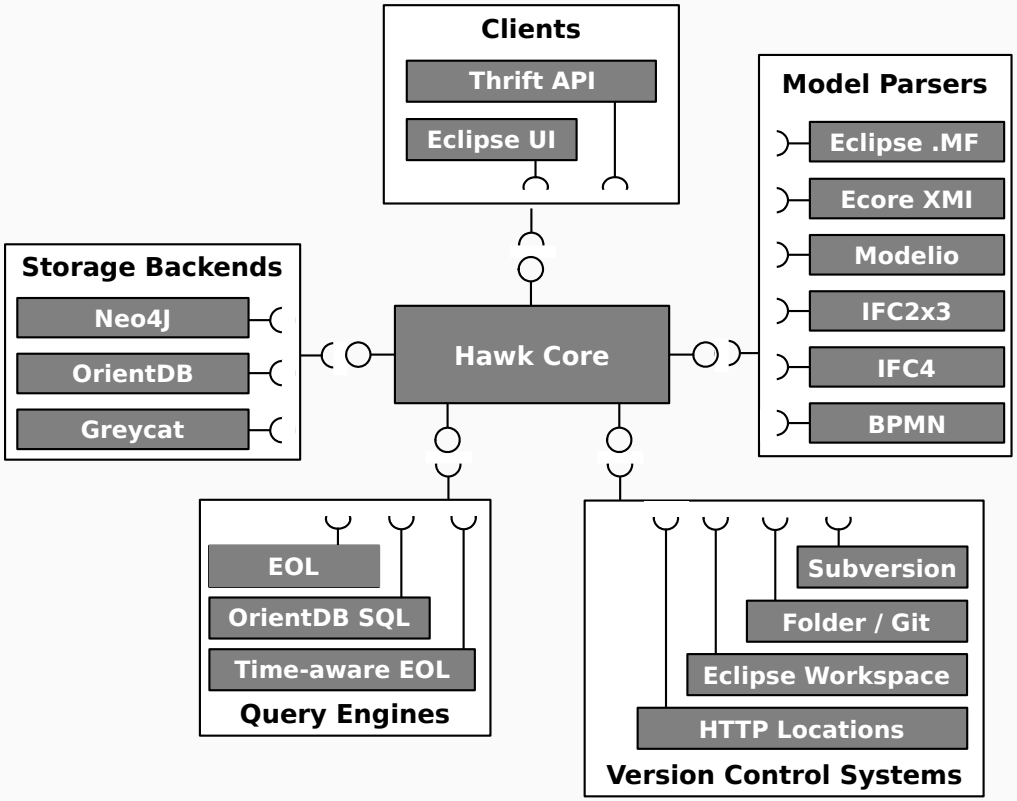
1. Hawk watches VCS, indexes
2. User queries Hawk over WS
3. Hawk runs query through NoSQL database efficiently
4. Hawk replies with result

Deployment



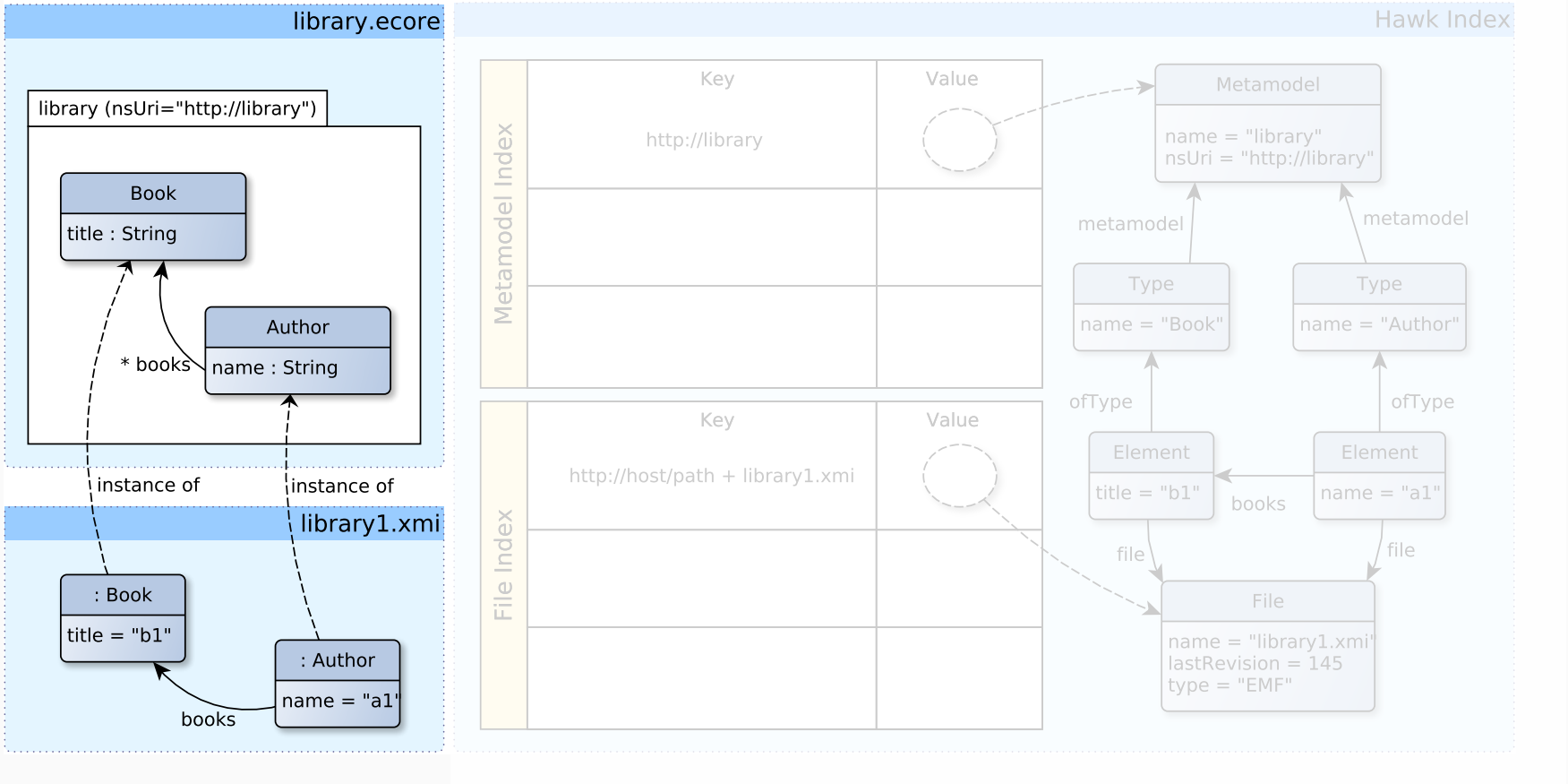
- Hawk can run as Eclipse plug-in, Java library, or network service
- We can have it watch over various types of locations:
  - Version control systems (SVN/Git repositories)
  - File stores (local folders, Eclipse workspaces, HTTP locations)

Component-based architecture



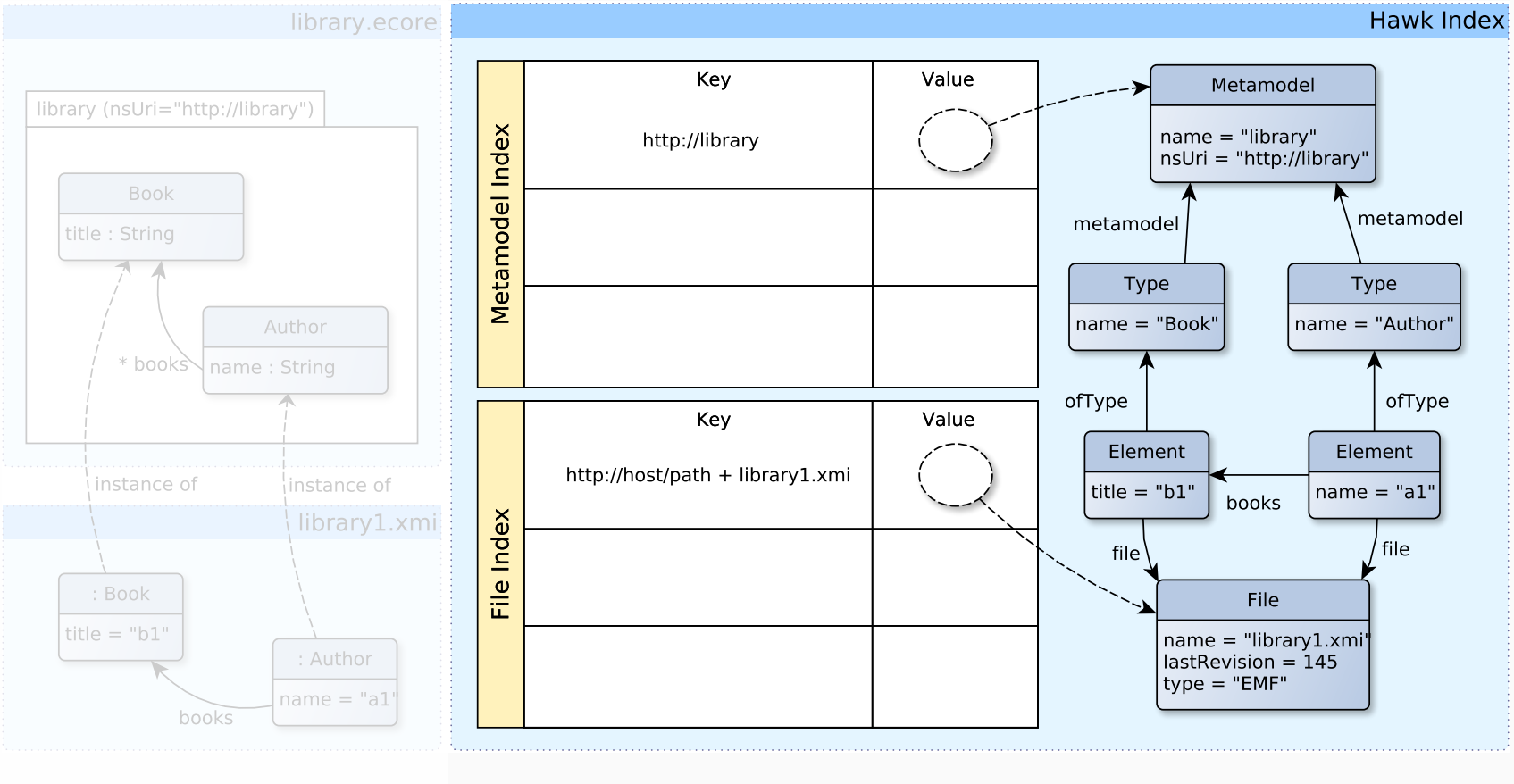
- Core: incremental graph updating + component interfaces
- Backends: Neo4j (fastest), OrientDB (multi-master), Greycat
- Clients: Eclipse GUI, cross-language Apache Thrift web services
- Query engines: Epsilon Object/Pattern Languages, OrientDB SQL
- Model parsers: EMF/Modelio models, Eclipse plug-in manifests...

Example for a library model



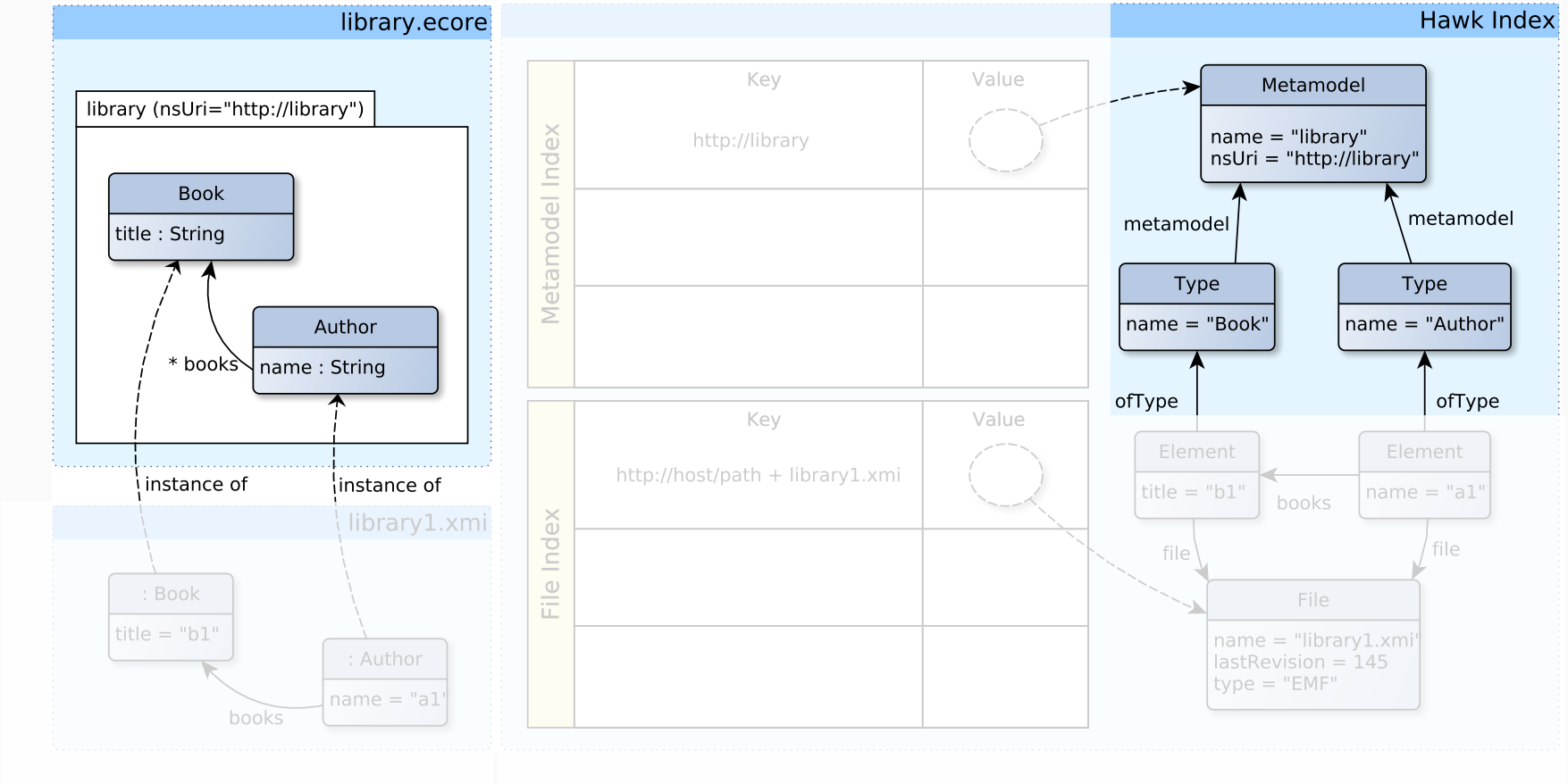
We go from these model files...

Example for a library model



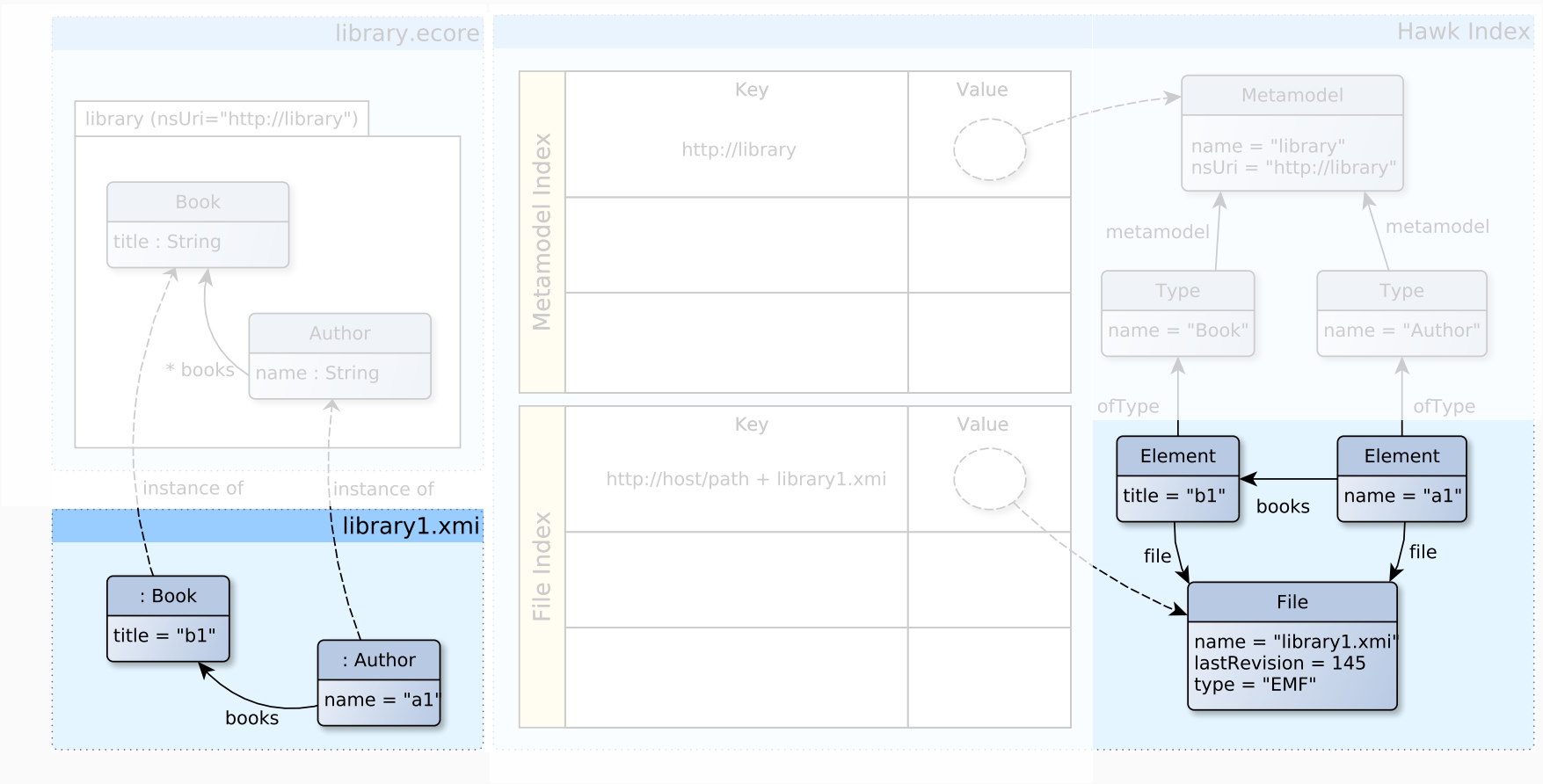
... to these NoSQL graphs.

Example for a library model



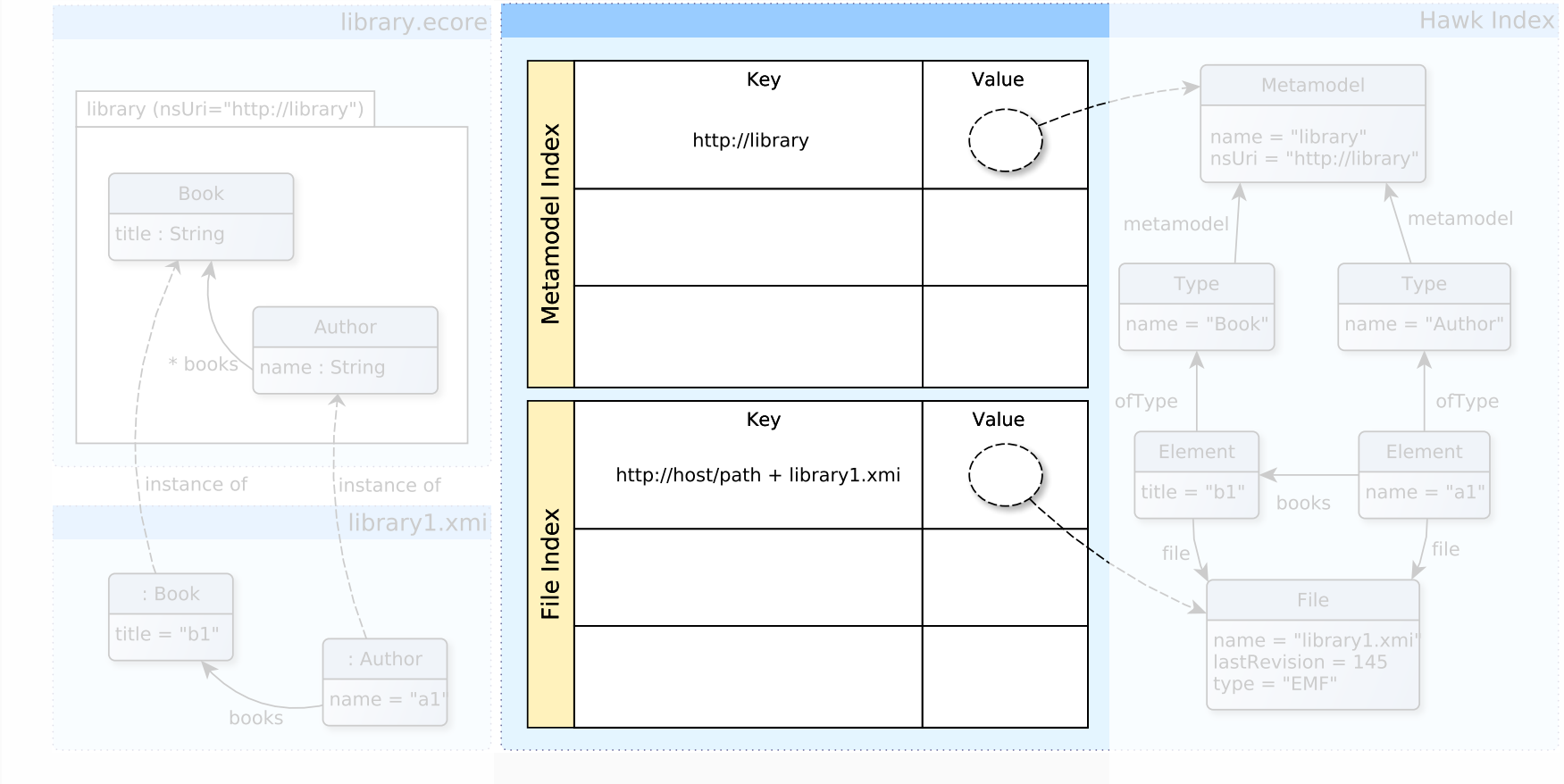
- Ecore packages → metamodel nodes
- Ecore classes → type nodes

Example for a library model



- Physical files → file nodes
- Model elements → element nodes

Example for a library model

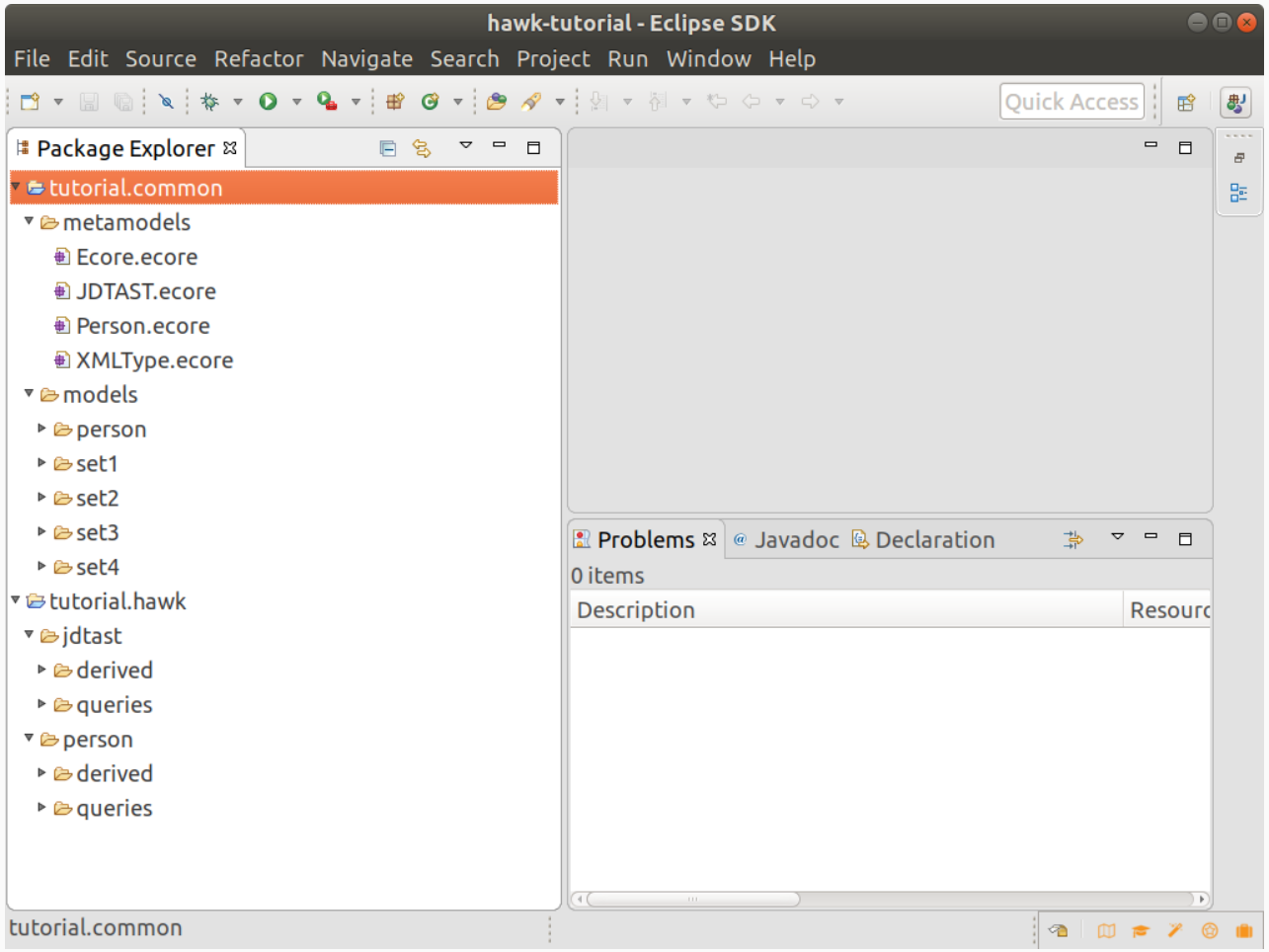


- MM index: package URI $\rightarrow$ metamodel node
- File index: file path $\rightarrow$ file node
- Users can define custom indices by attribute/expression

Hands-on time!

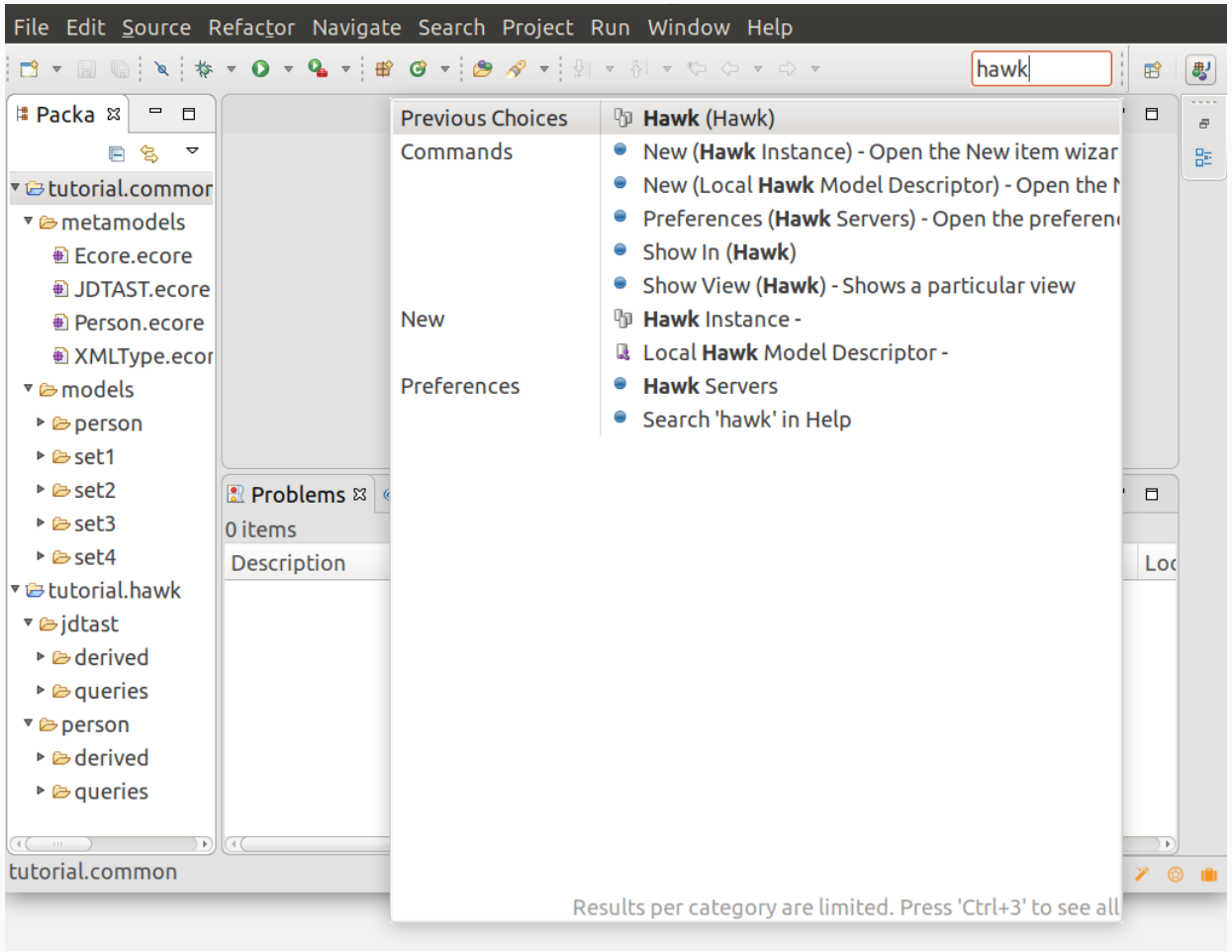
Let's index a Java model and find singletons.

Preparing the workspace

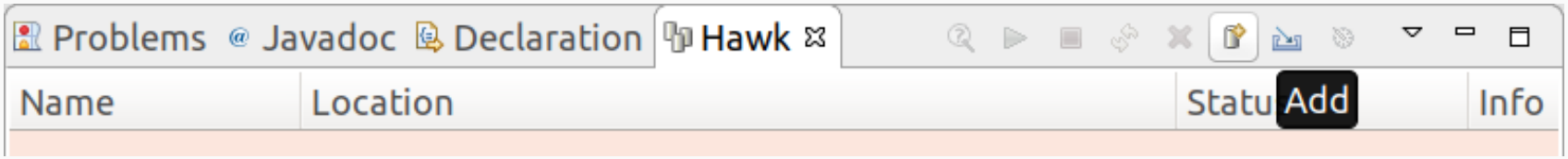


Open Eclipse and import `tutorial.common` and `tutorial.hawk`.

Opening the main Hawk view

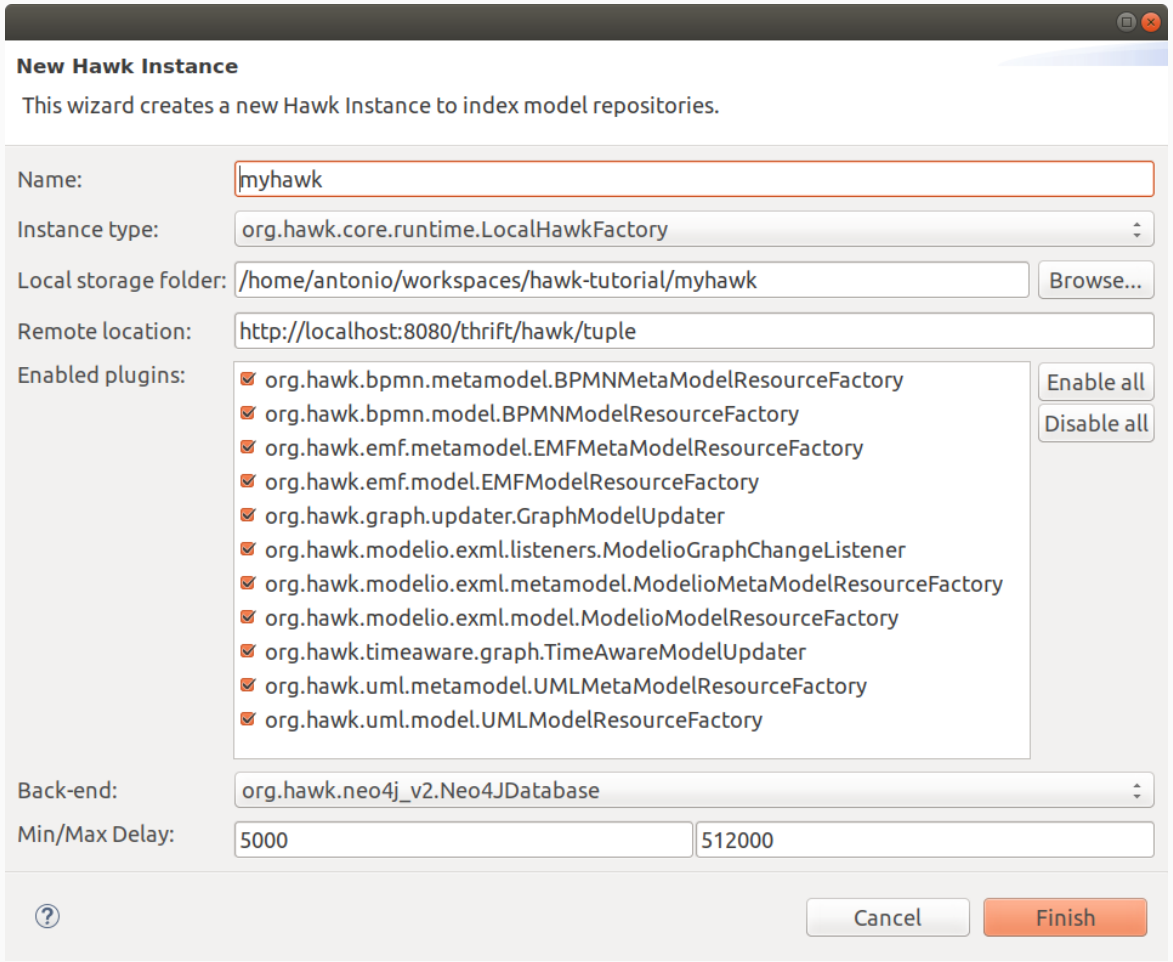


Adding a new Hawk index



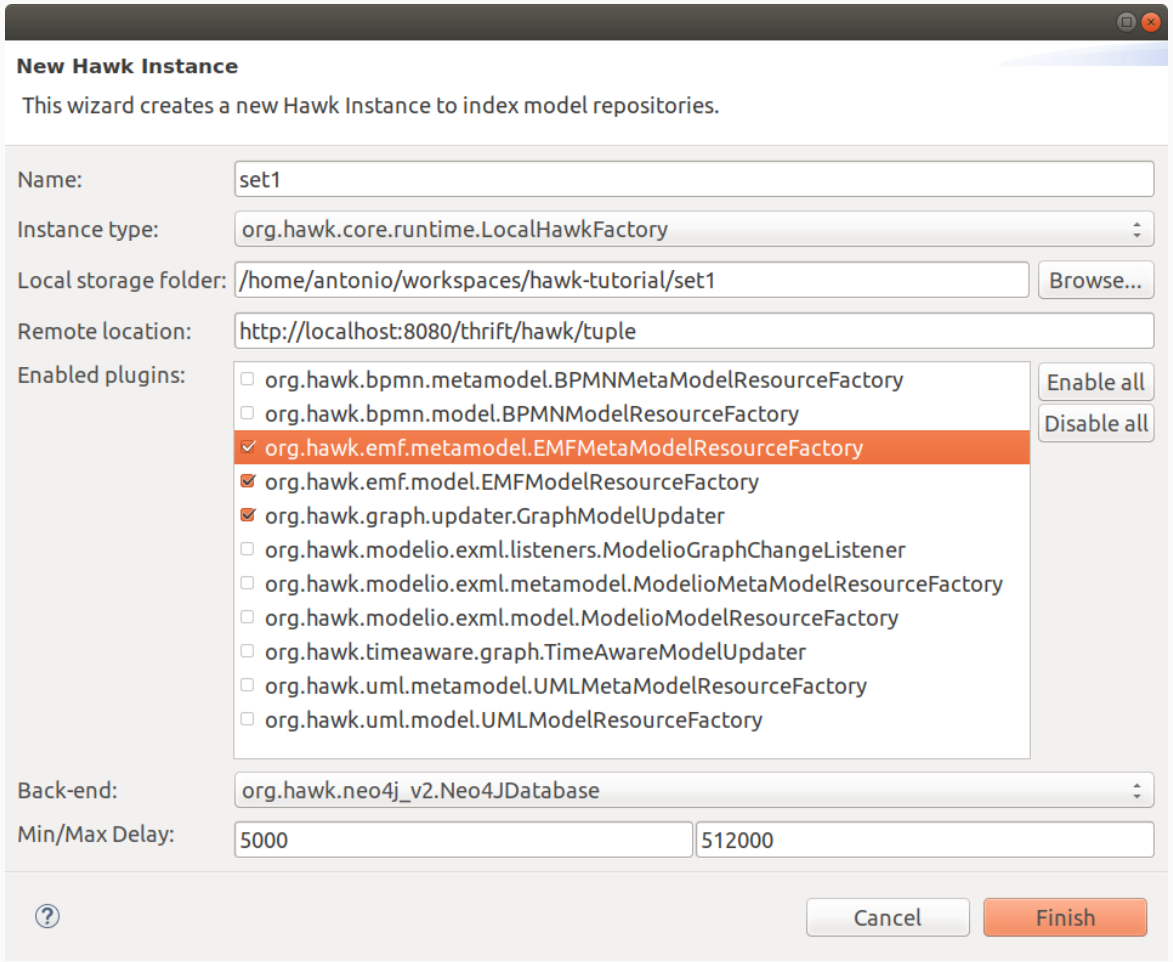
Click on the "Add" button. We can keep as many different indices as we want within the same Eclipse installation.

Options, options!



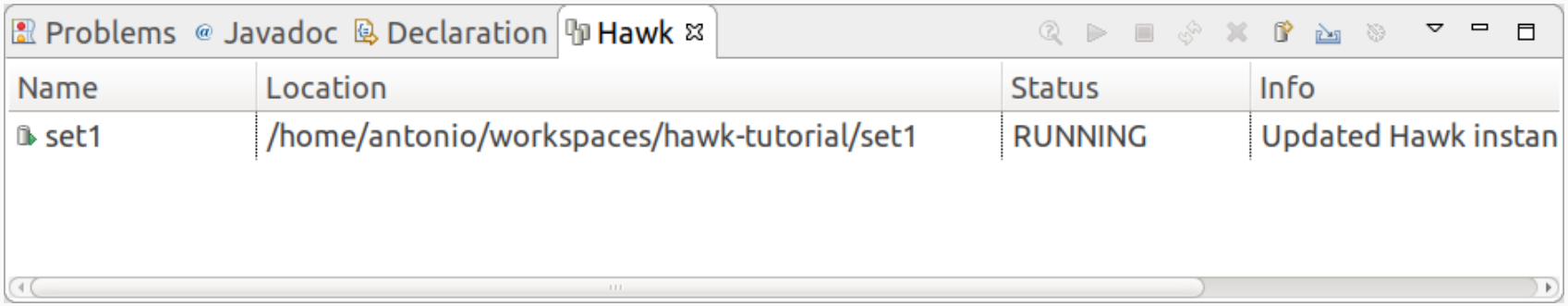
There are many options here: we can use a local or a remote Hawk, we can choose the backend, and we can enable/disable plugins at will.

Options for the JDTAST models



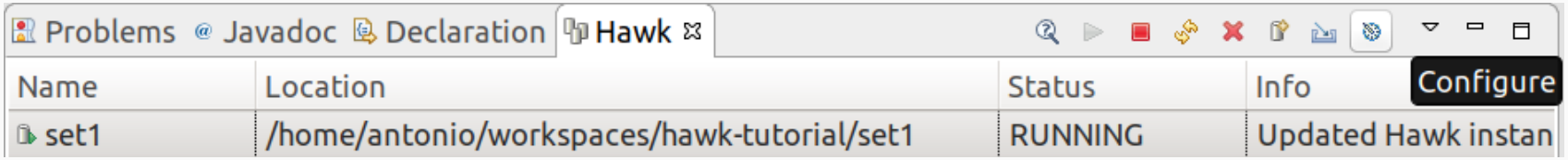
We will go with a local, Neo4j-based, EMF-focused index.

Index created



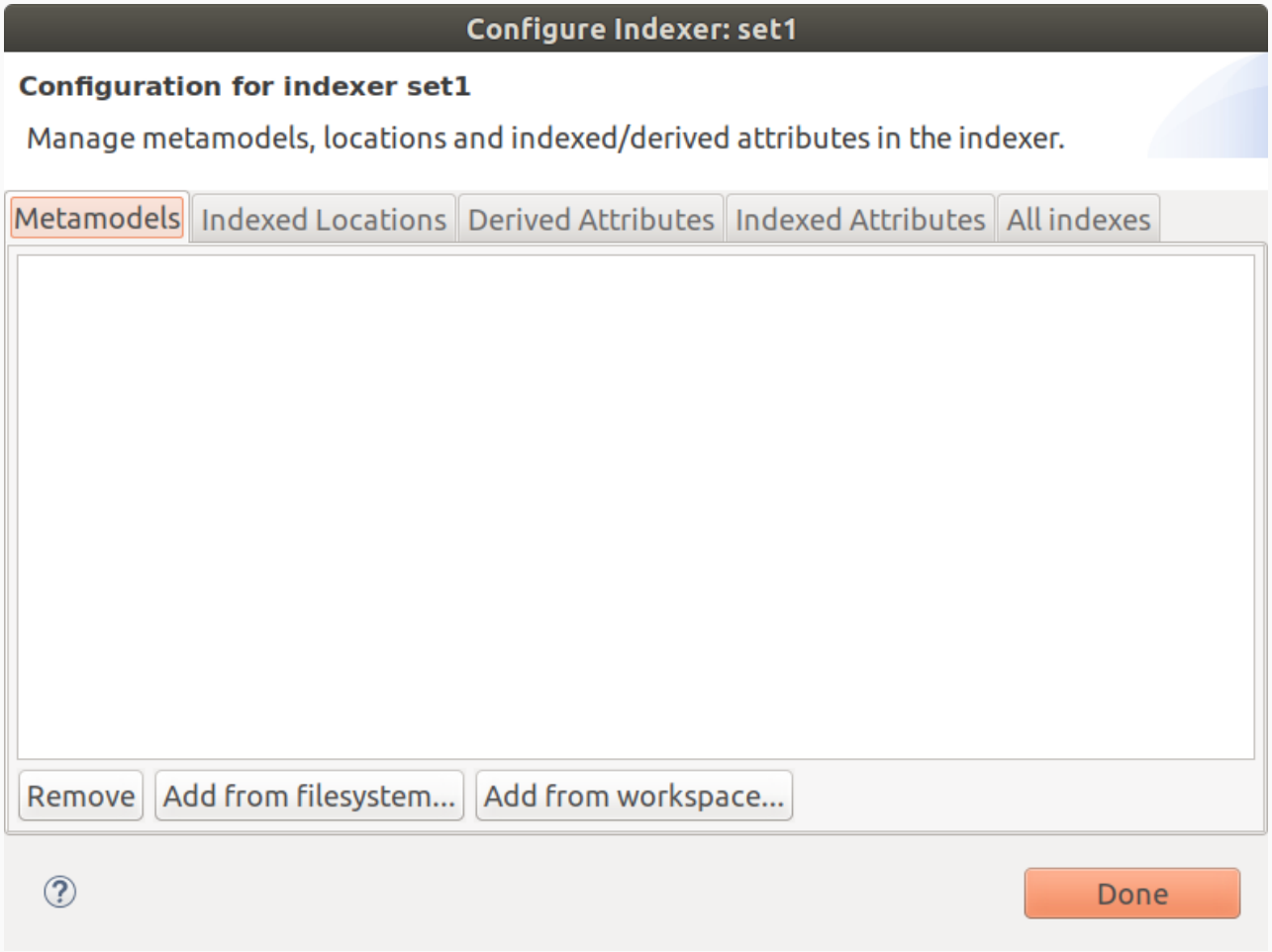
We have created an empty index. Time to configure it.

Configuring the index



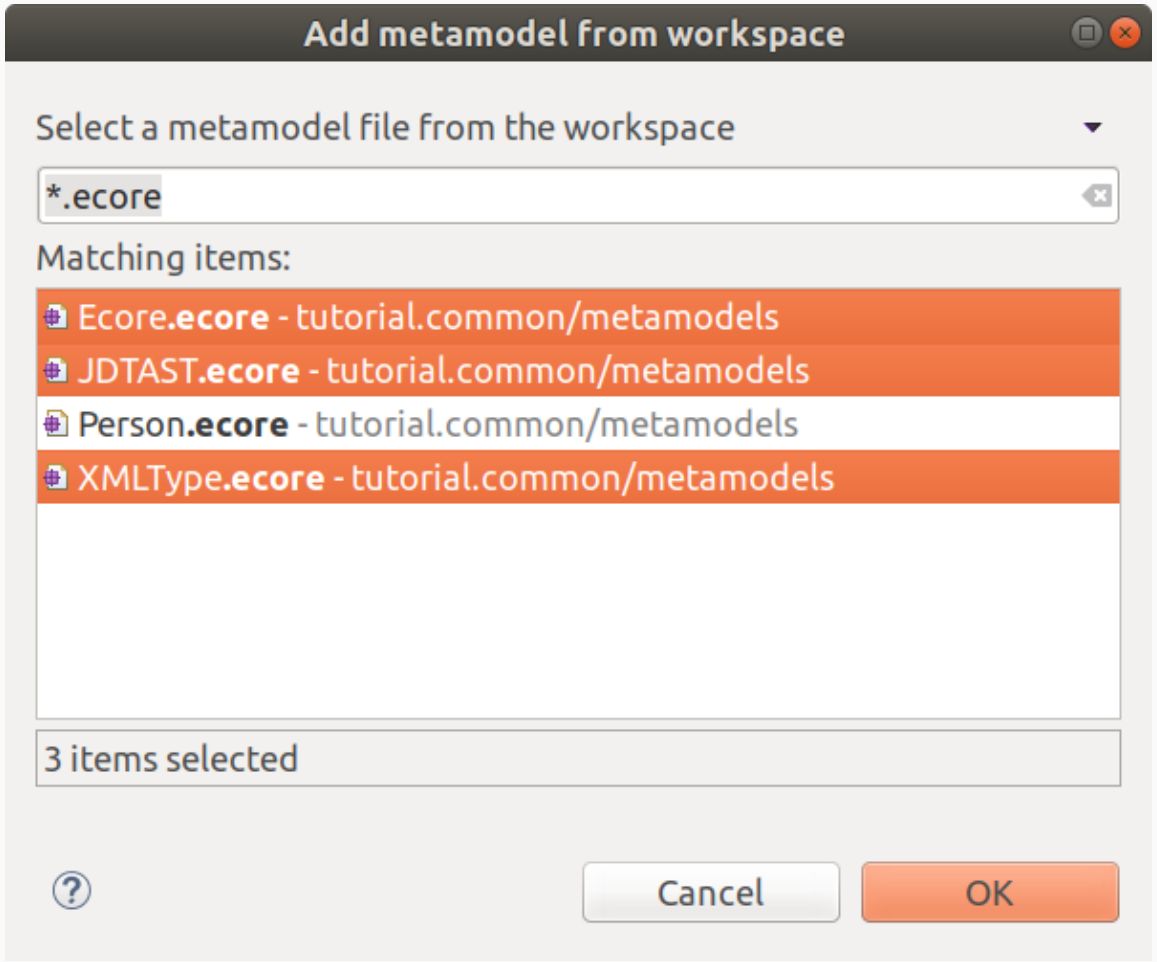
Let's click on the "Configure" button. This will open a new dialog.

Configure dialog



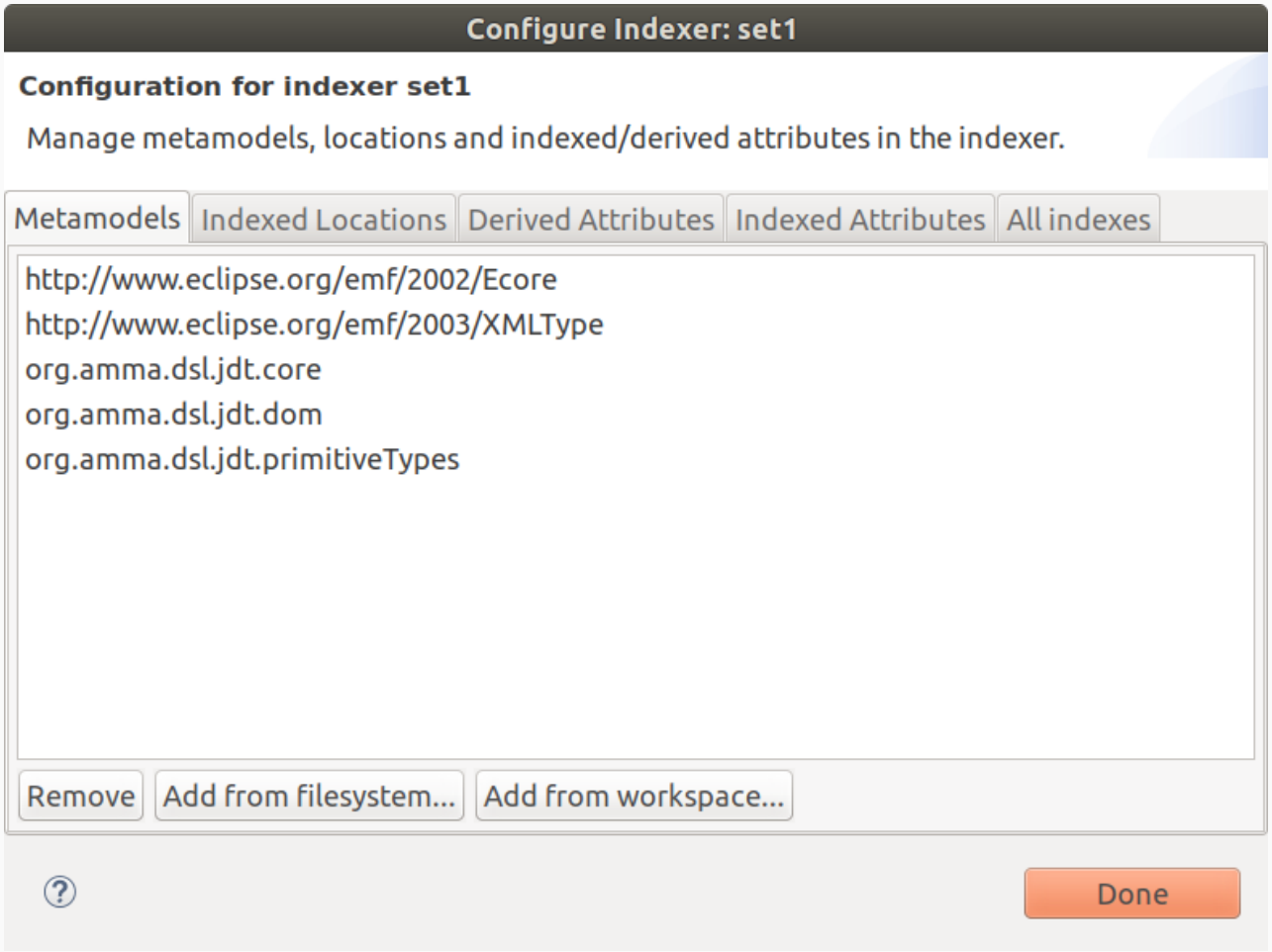
The dialog is divided into tabs for managing metamodels, locations to index, and indexed/derived properties.

Adding metamodels



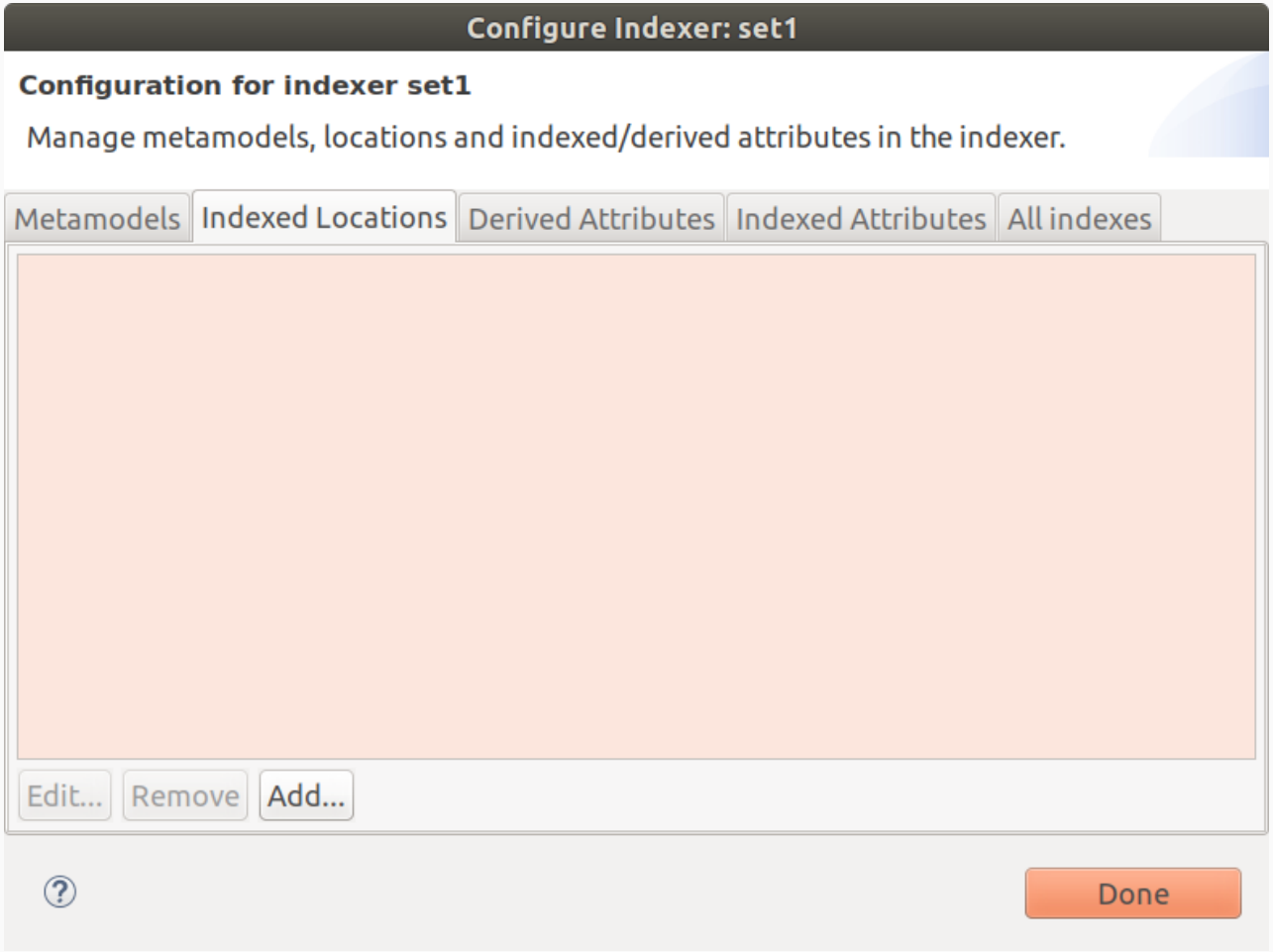
Click on "Add from workspace" and select the Ecore, XMLTypes and JDTAST metamodels. These will be turned into Neo4j nodes and edges.

Added metamodels



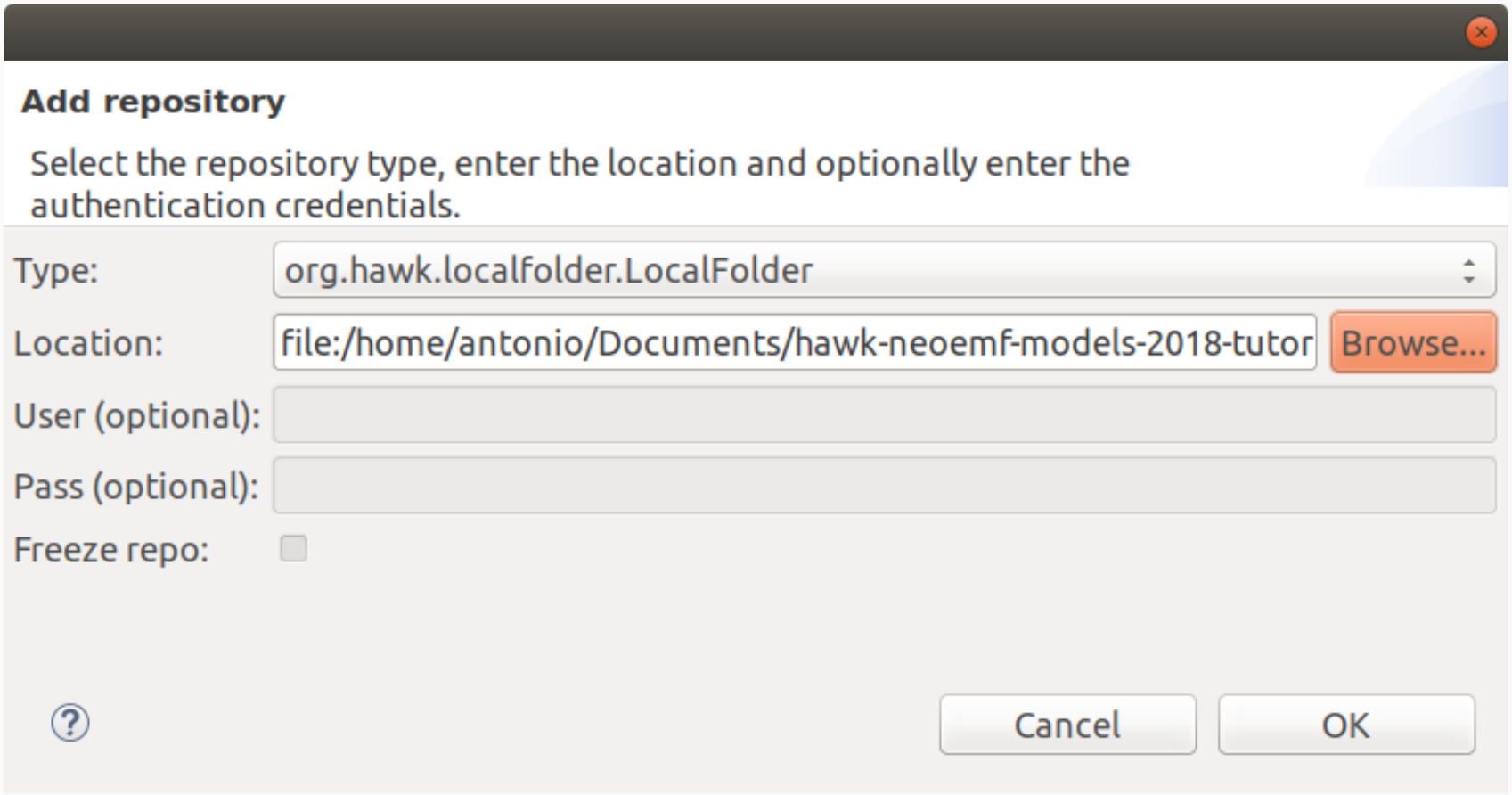
The metamodels have been registered: we can index models now.

Add locations



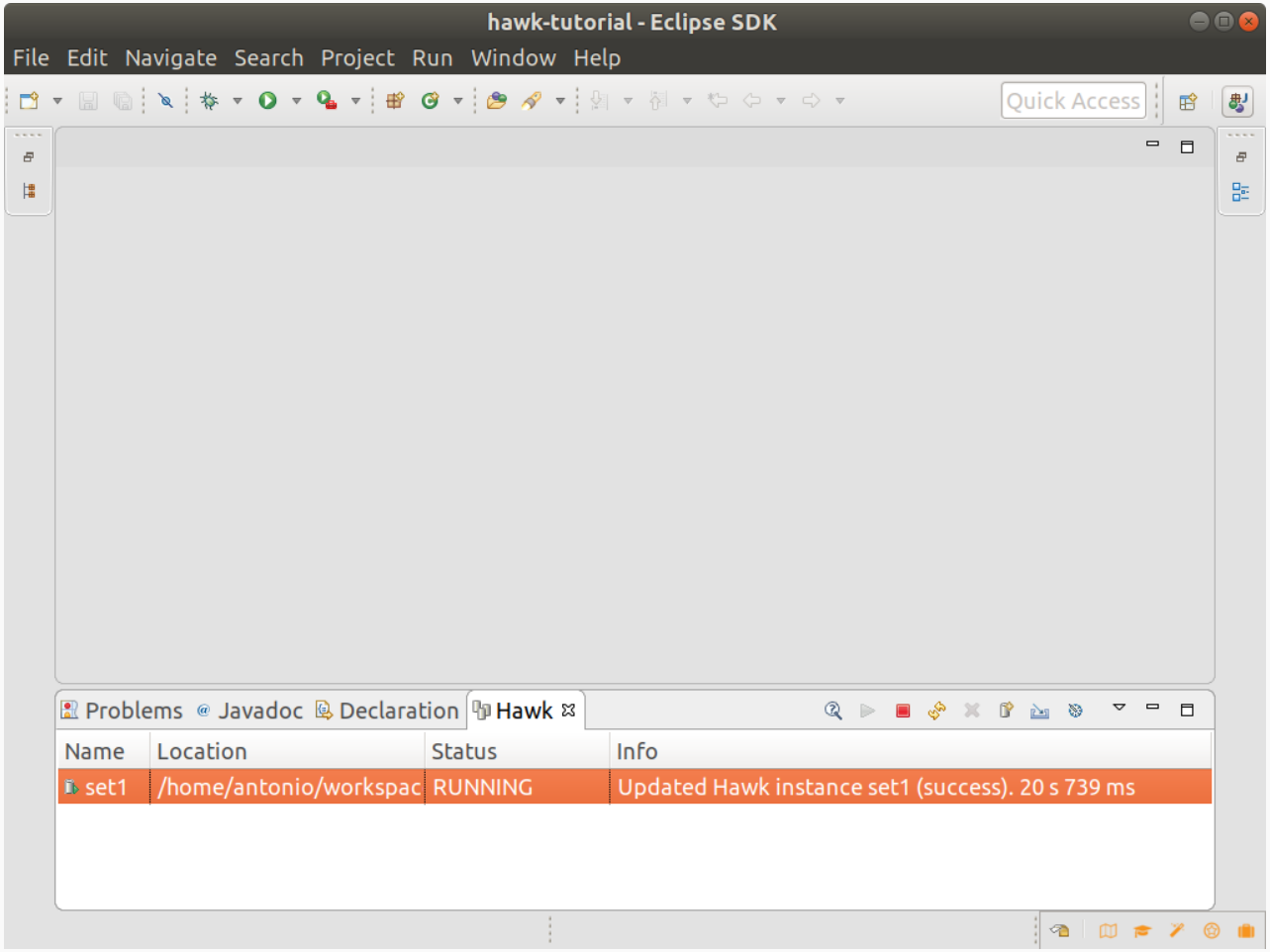
Go to the “Indexed Locations” tab and click on “Add”.

Adding a local folder



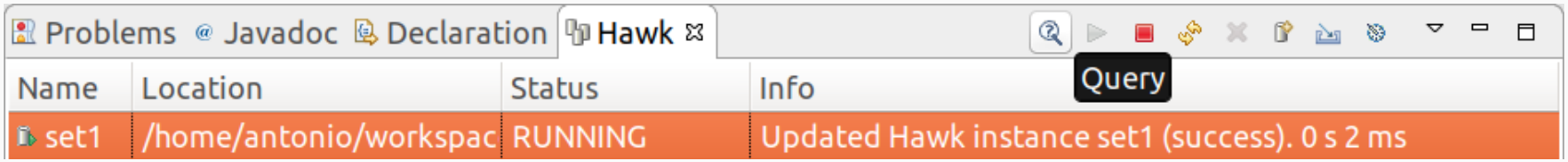
We want to index the "LocalFolder" in the `tutorial.common/models/set1` folder.

Added local folder



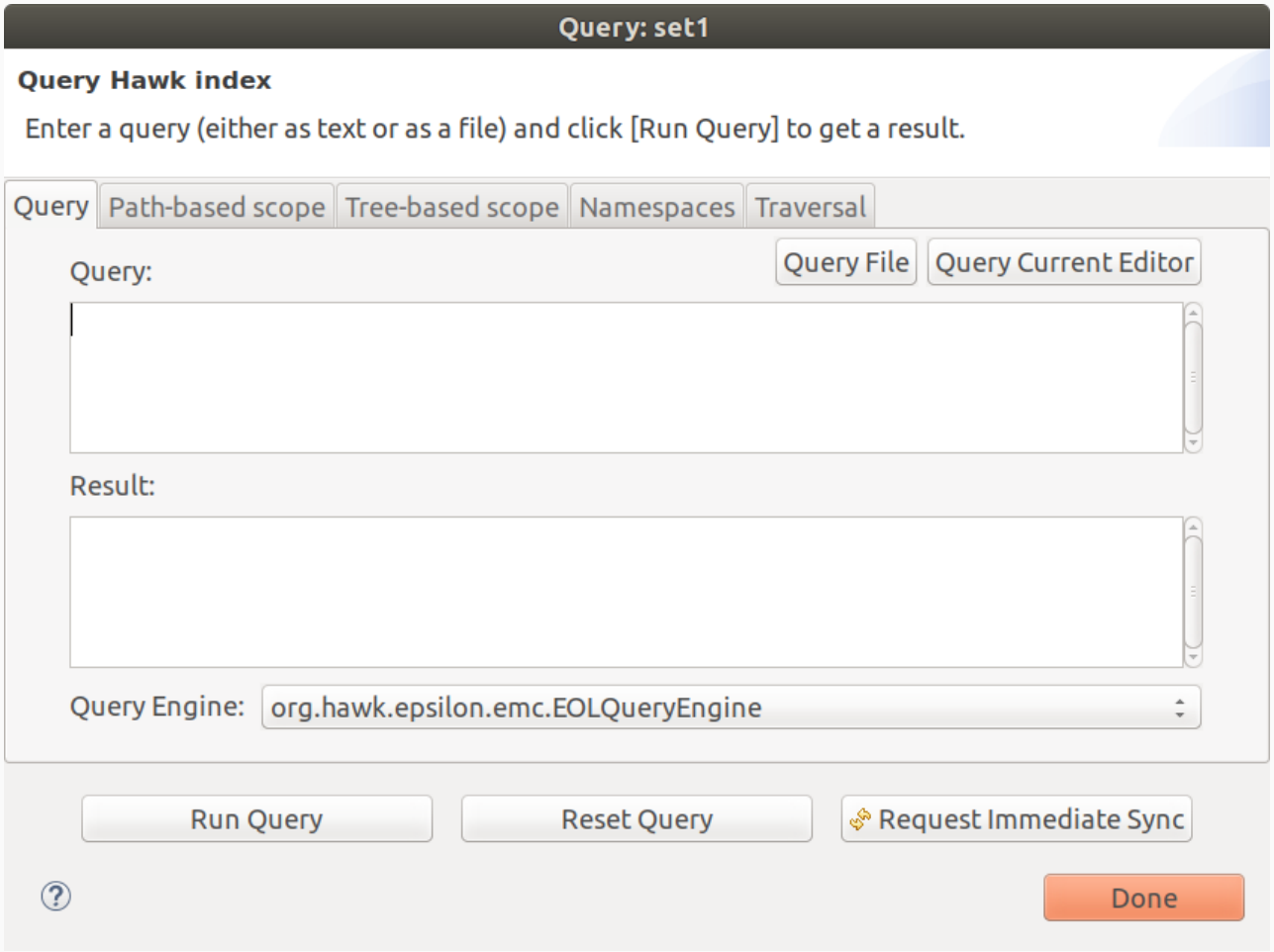
Wait a bit, watching the "Status" column, and the models will be indexed. Once it's done, the index is ready to be queried.

Query button



Click on the “Query” button.

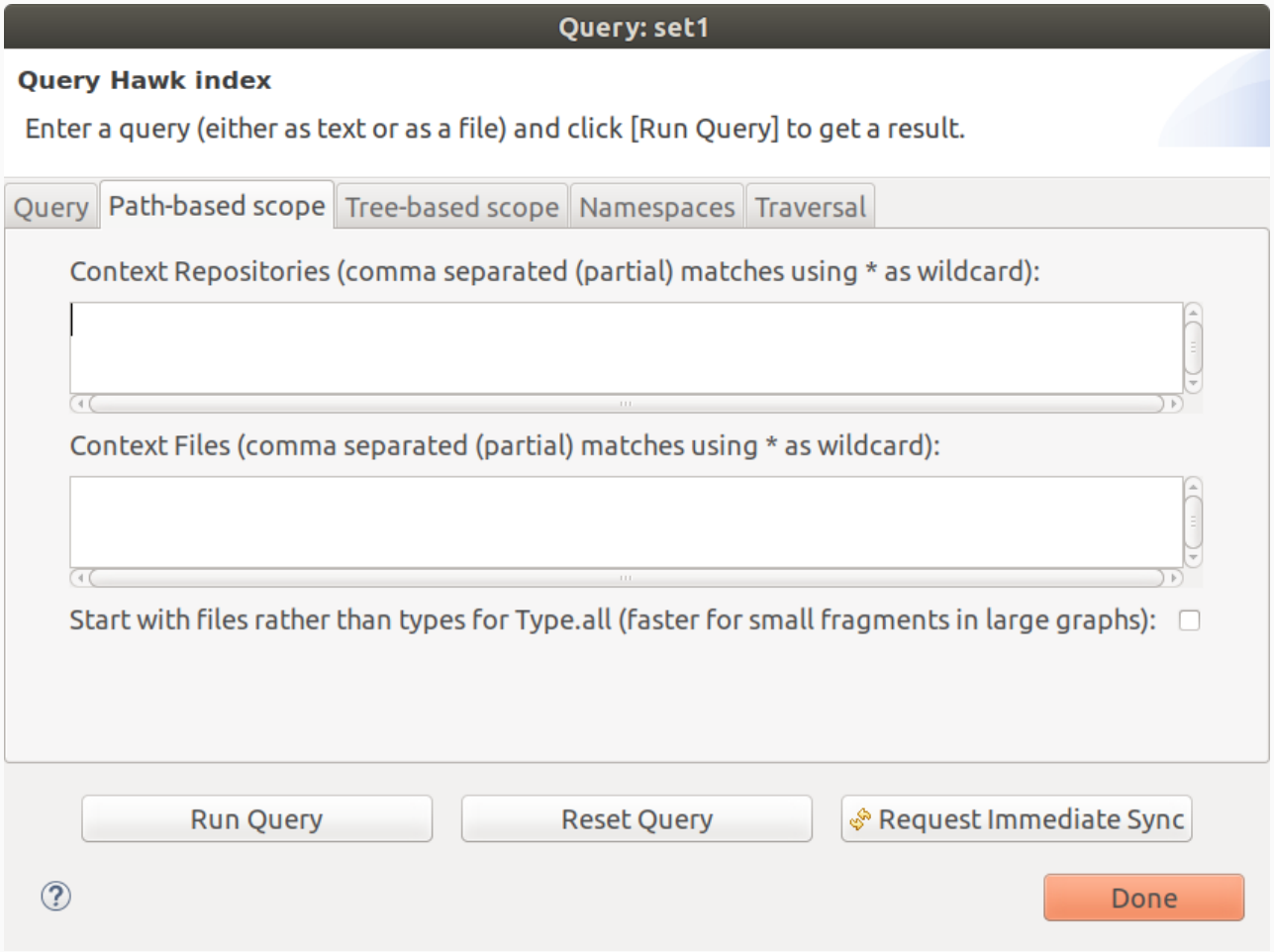
Query dialog: main tab



The "Query" dialog is quite involved: let's go bit by bit.

The main page allows us to enter the query by hand, from a file or from the current editor, run/stop/reset it, and request manual syncs.

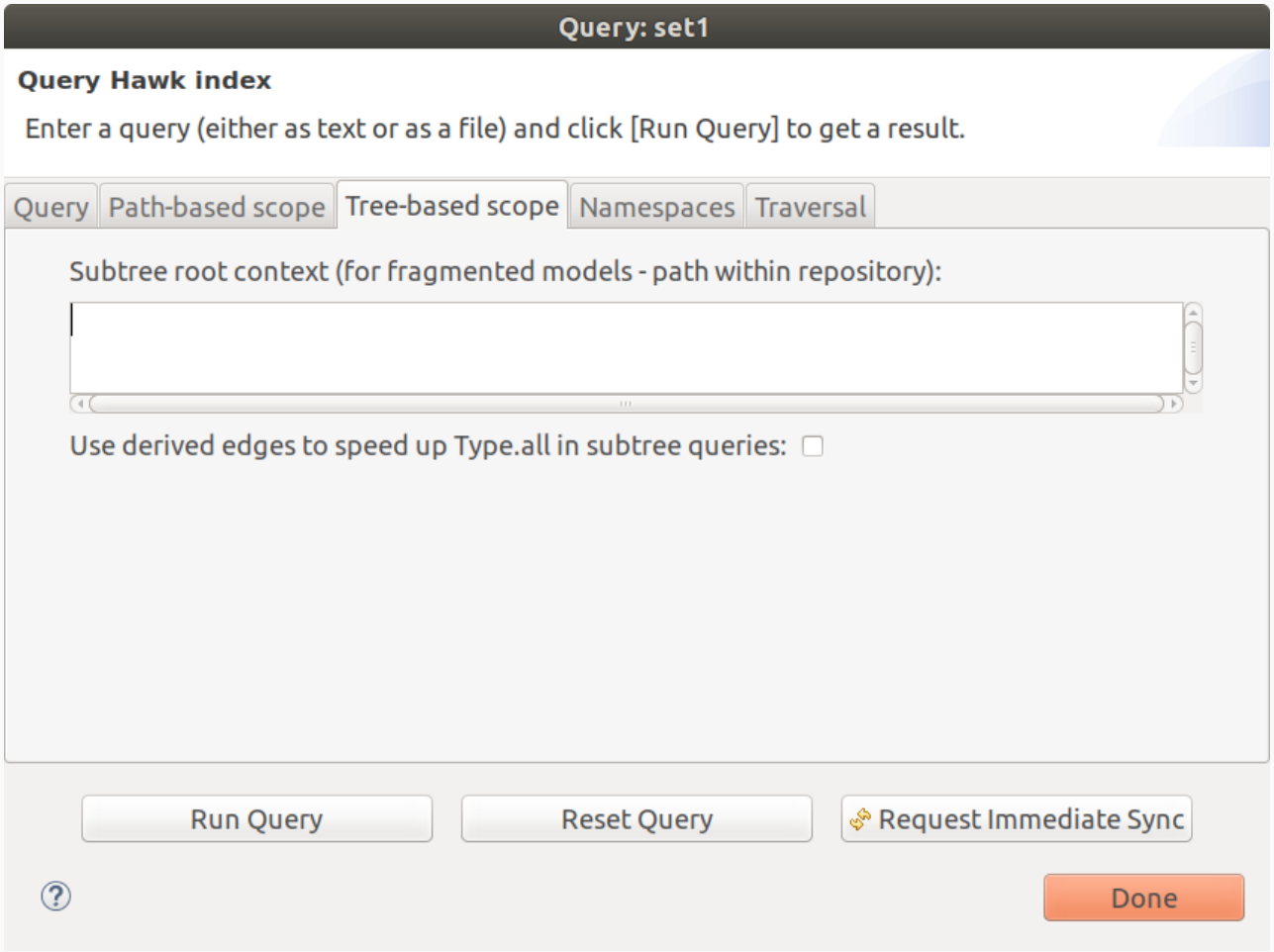
Query dialog: path-based scope



We could optionally limit the scope of the query to specific repositories and/or files. This can be useful when querying large collections of models.

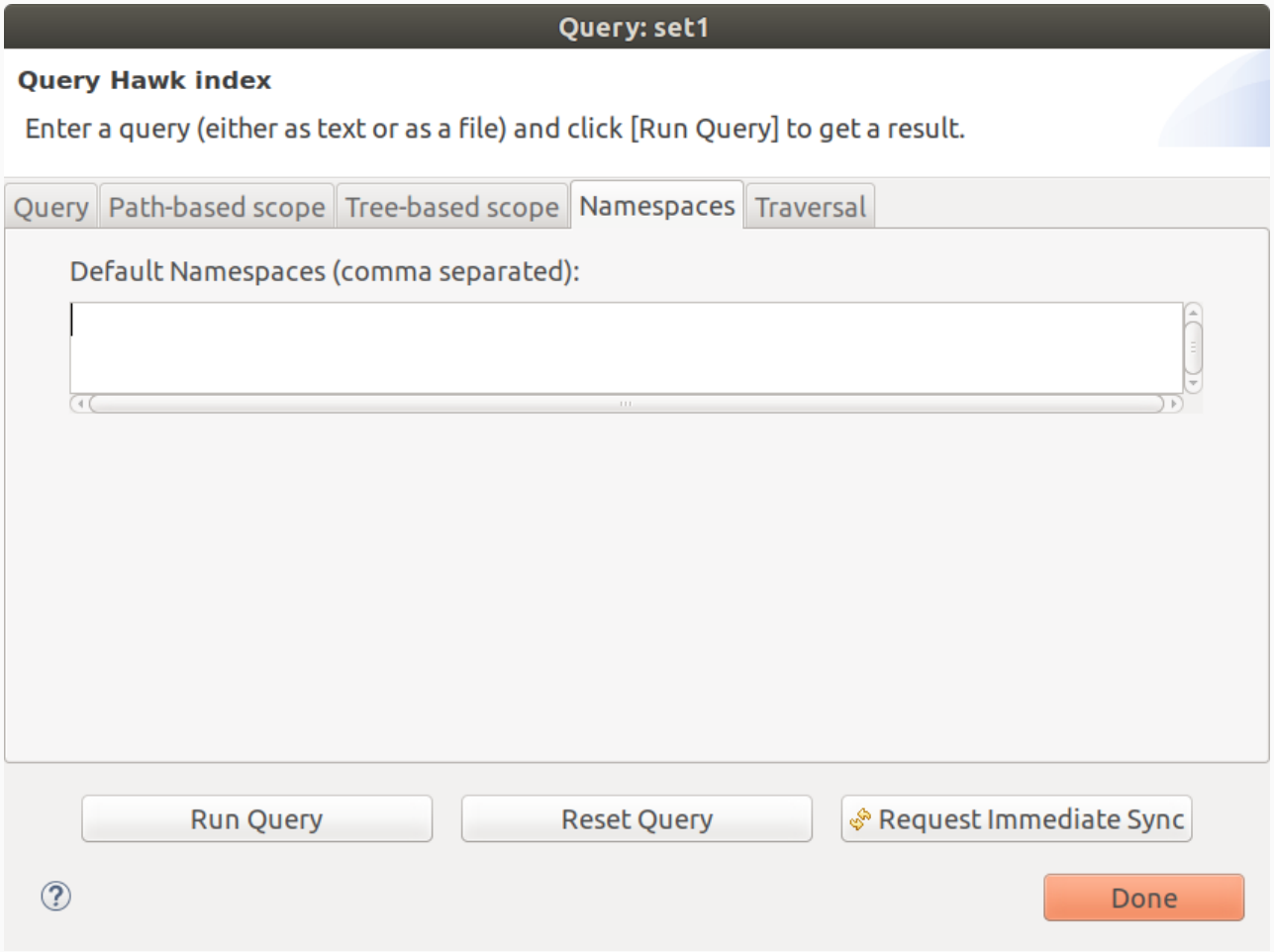
Patterns are essentially glob-style: e.g. `folder/f*.xmi`.

Query dialog: subtree-based scope



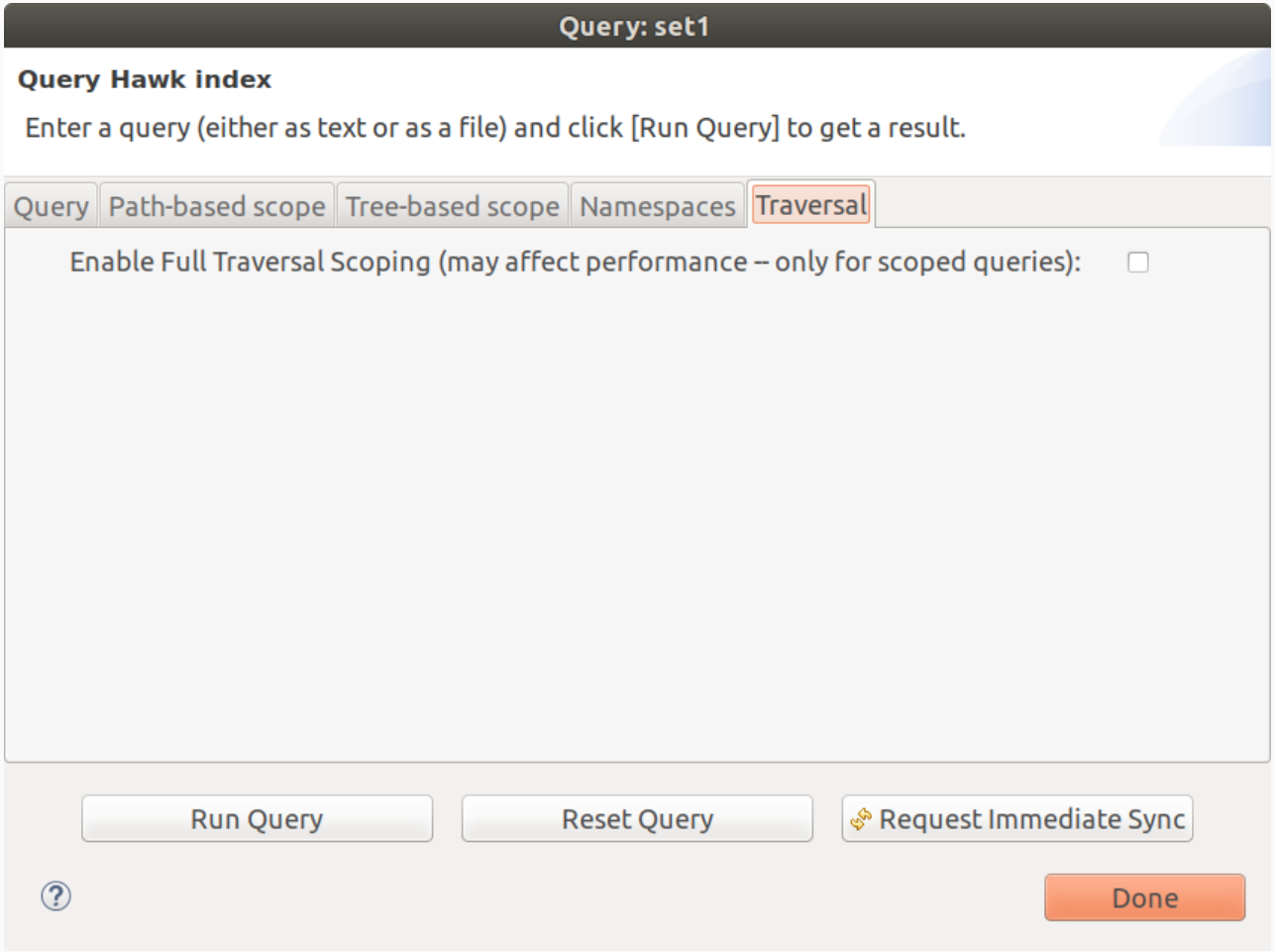
Another option is to limit scope to the objects contained within a certain top-level file. This can be useful when querying hierarchically fragmented models (e.g. those created with EMF-Splitter by Antonio Garmendia).

Query dialog: default namespaces



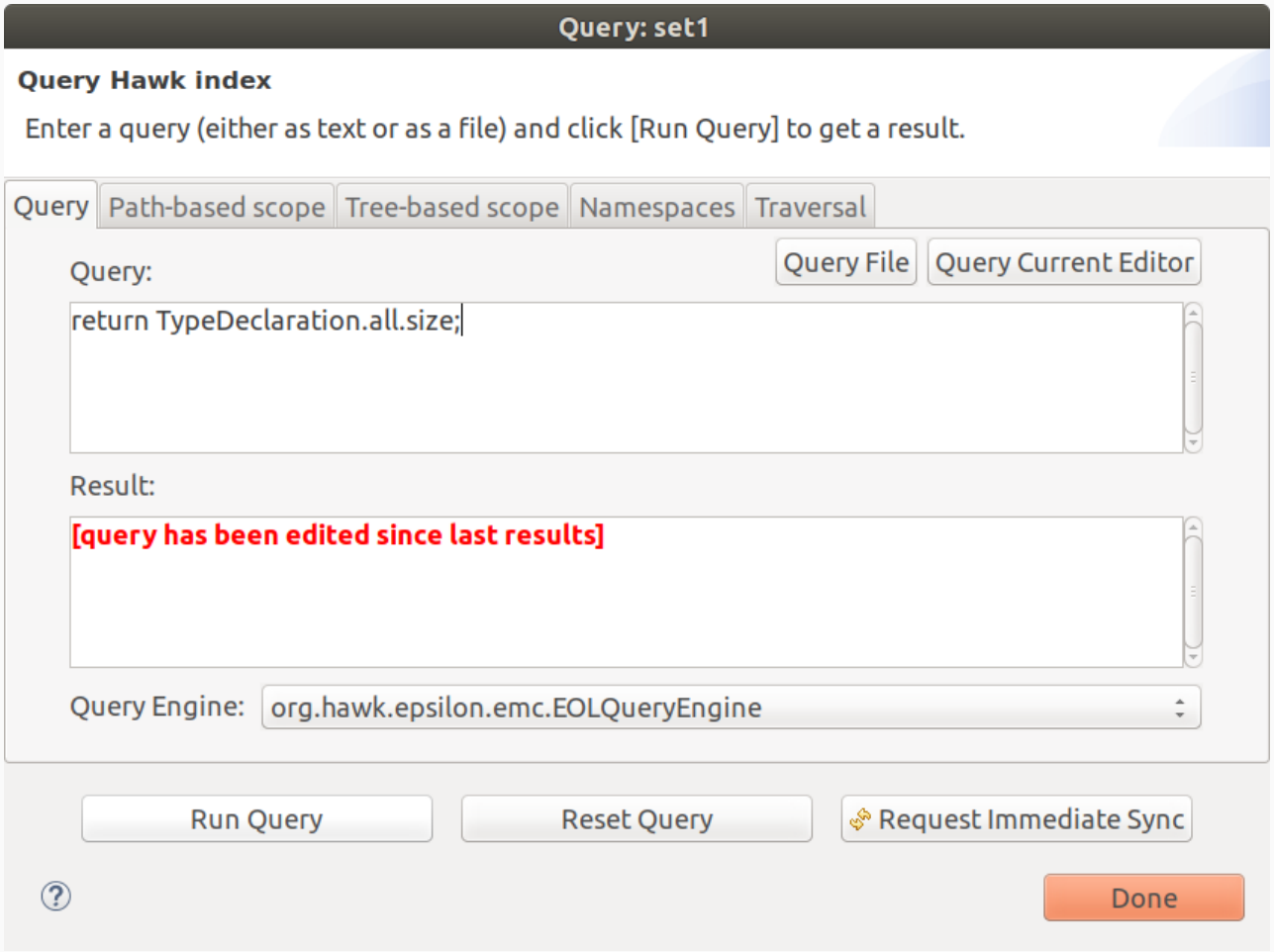
In some cases, "Type.all" might be ambiguous as we may have multiple types with the same unqualified name. Here we can specify which metamodel URIs should take precedence.

Query dialog: traversal scoping



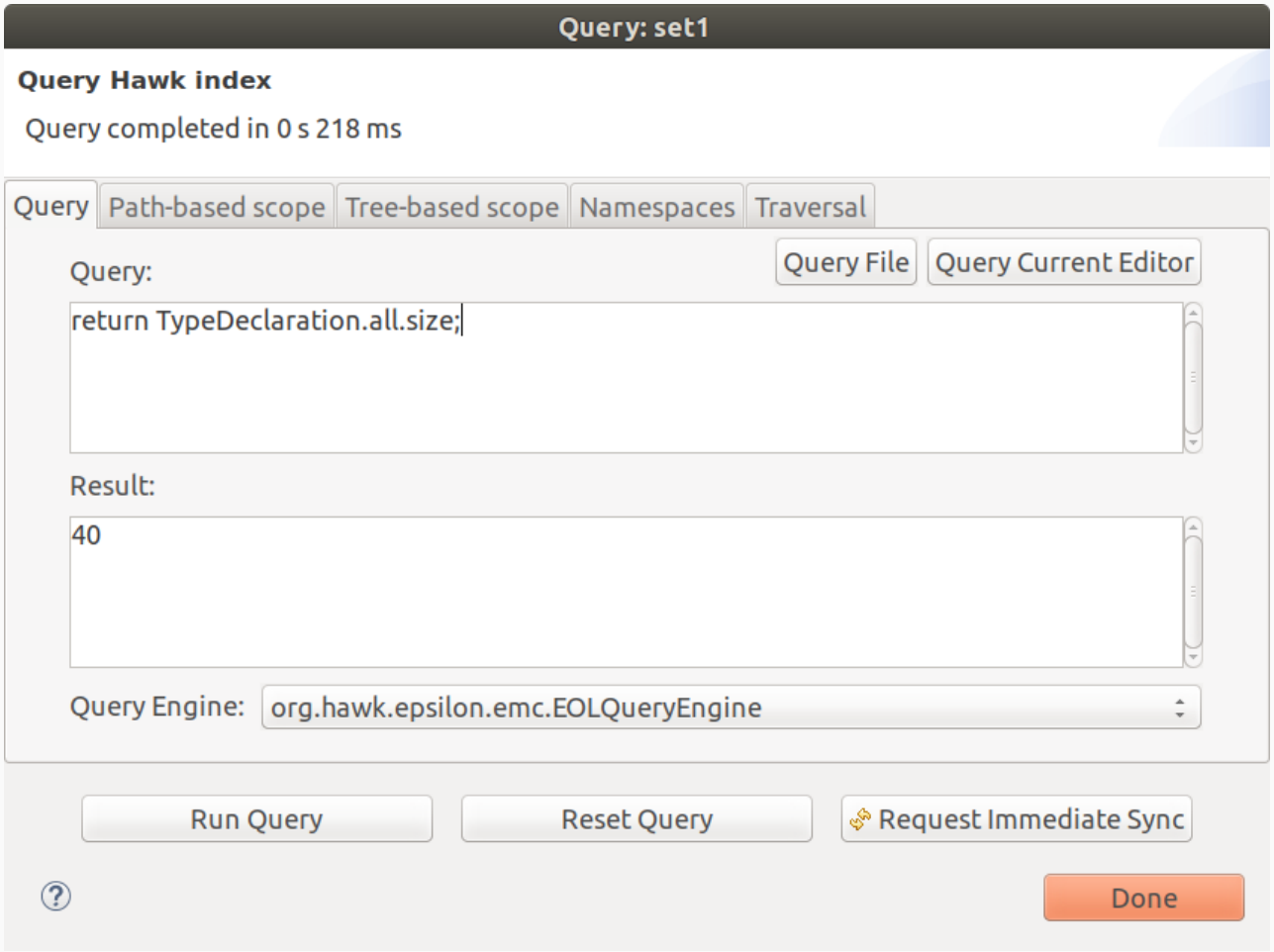
When following references from an object, we may want to ignore the ones that leave the path-/subtree-based scope that we defined before.

Query dialog: entering a query



Let's go back to the main tab and enter a simple query about the number of instances of a class. Click on "Run Query", which will temporarily change to "Stop Query".

Query dialog: query executed



The query finished running, and here we have the results.

Now that we know the basic use of Hawk, let's move to the more advanced features.

Indexed attributes

Finding a type by its name (`findByName.eol`)

```
return TypeDeclaration.all.selectOne(td |
  td.name.identifier = 'IdeInputFileObject');
```

This normally involves...

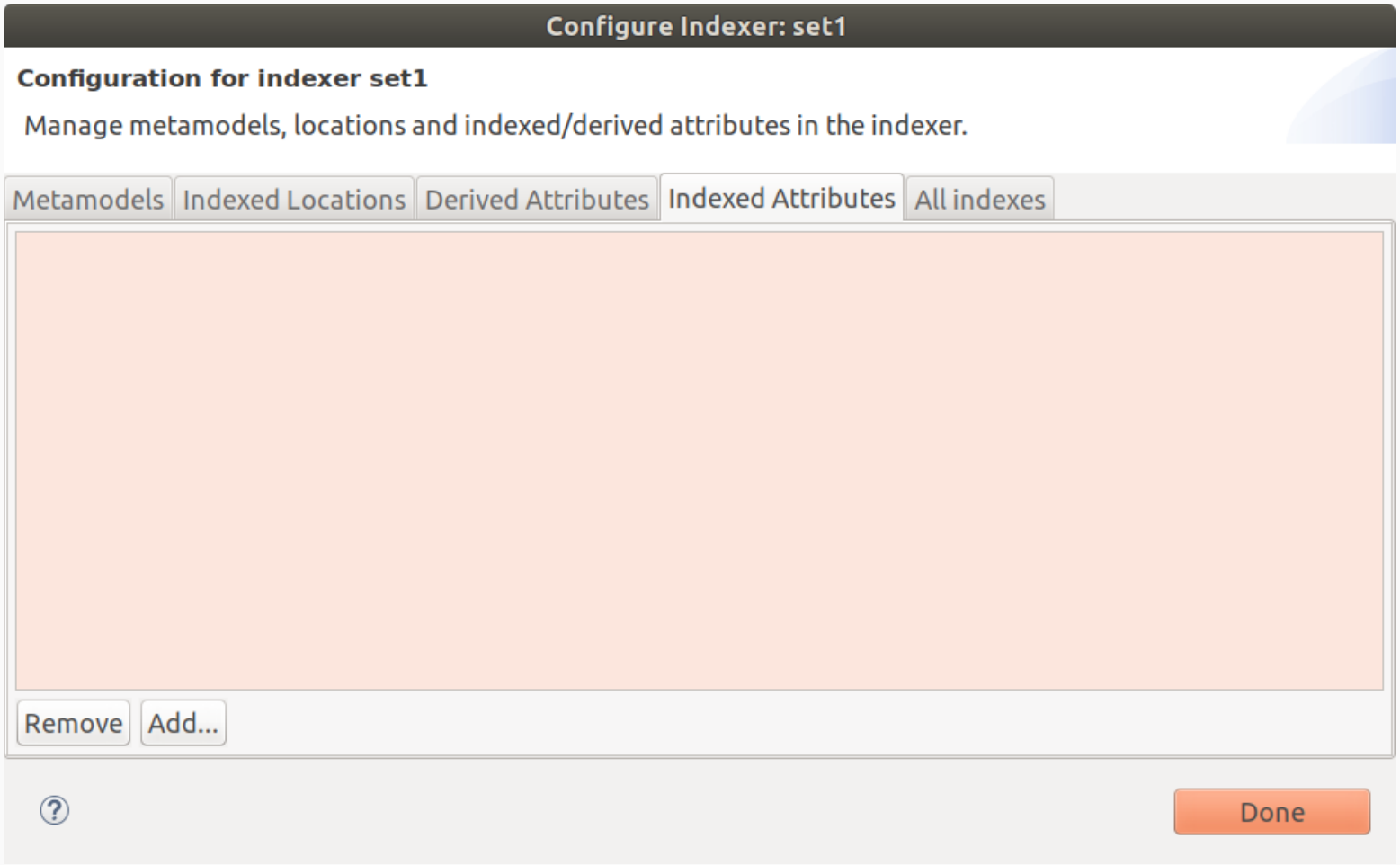
1. Iterating over all types
2. Following the "name" reference
3. Comparing the name

Replace with a lookup (`findByName-indexed.eol`)

We only need to tell it to index "SimpleName.identifier":

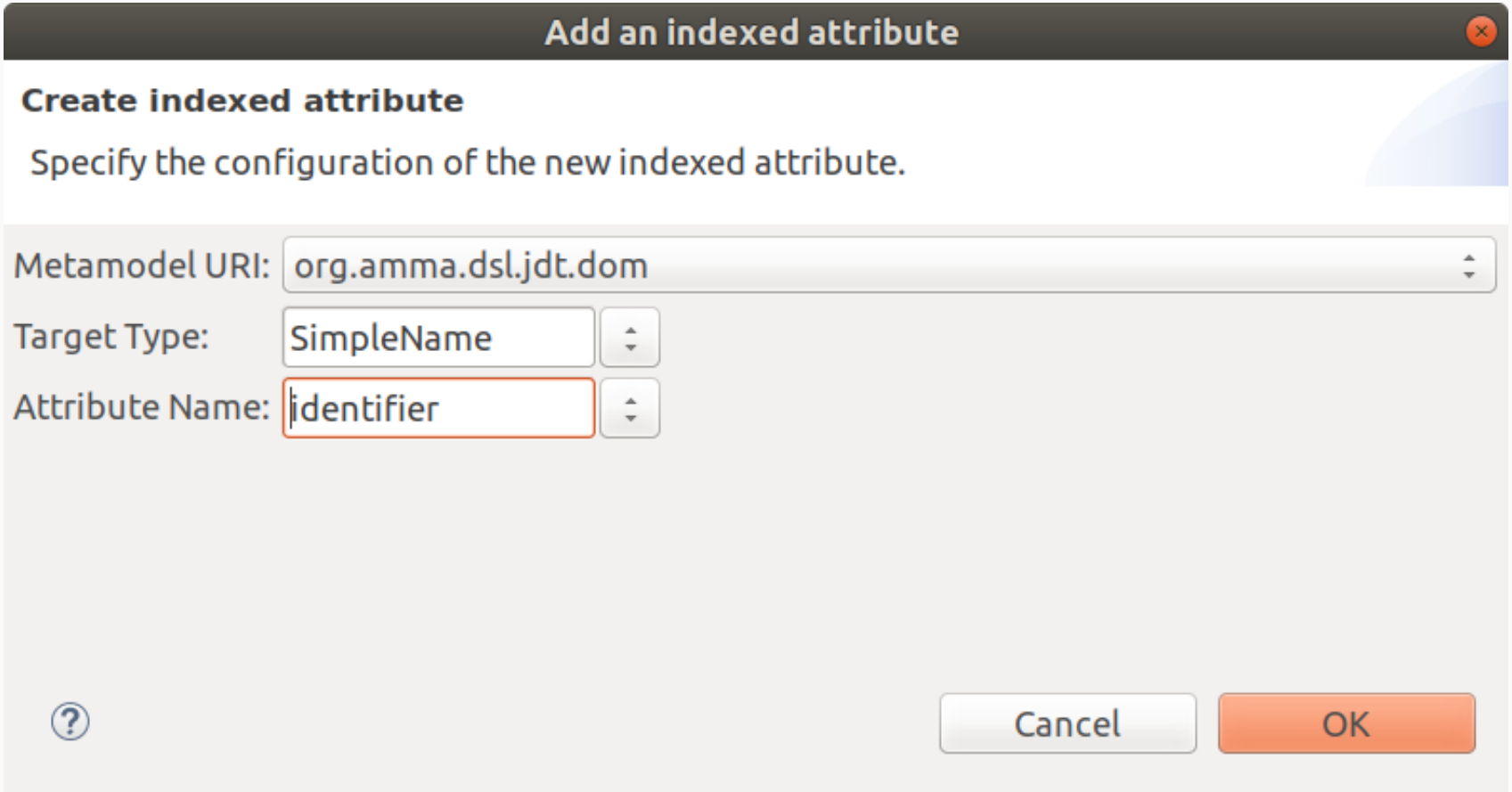
```
return SimpleName.all
  .select(sn | sn.identifier='IdeInputFileObject')
  .eContainer.select(c|c.isKindOf(TypeDeclaration));
```

Indexed attributes tab



Go back to the “Configure” tab and select the “Indexed Attributes” tab. Click on “Add”, and a new dialog should pop up.

Options for the indexed attribute



Use the combo boxes to pick the metamodel, type and attribute shown above. Click on “Add”.

Derived attributes

Original query for finding singletons (`singletons.eol`)

```
return TypeDeclaration.all.select(td | td.bodyDeclarations.exists(
  md:MethodDeclaration | md.returnType.isTypeOf(SimpleType)
    and md.returnType.name.fullyQualifiedName
      = td.name.fullyQualifiedName
    and md.modifiers.exists(mod:Modifier | mod.public = true)
    and md.modifiers.exists(mod:Modifier | mod.static = true)
));
```

Can we do it faster?

- Checking if a method is public or static requires traversing references
- Same goes for checking if it returns an instance of itself
- In Hawk, we can precompute this
- When files change, only the affected values are recomputed

Use of derived attributes as precomputed values

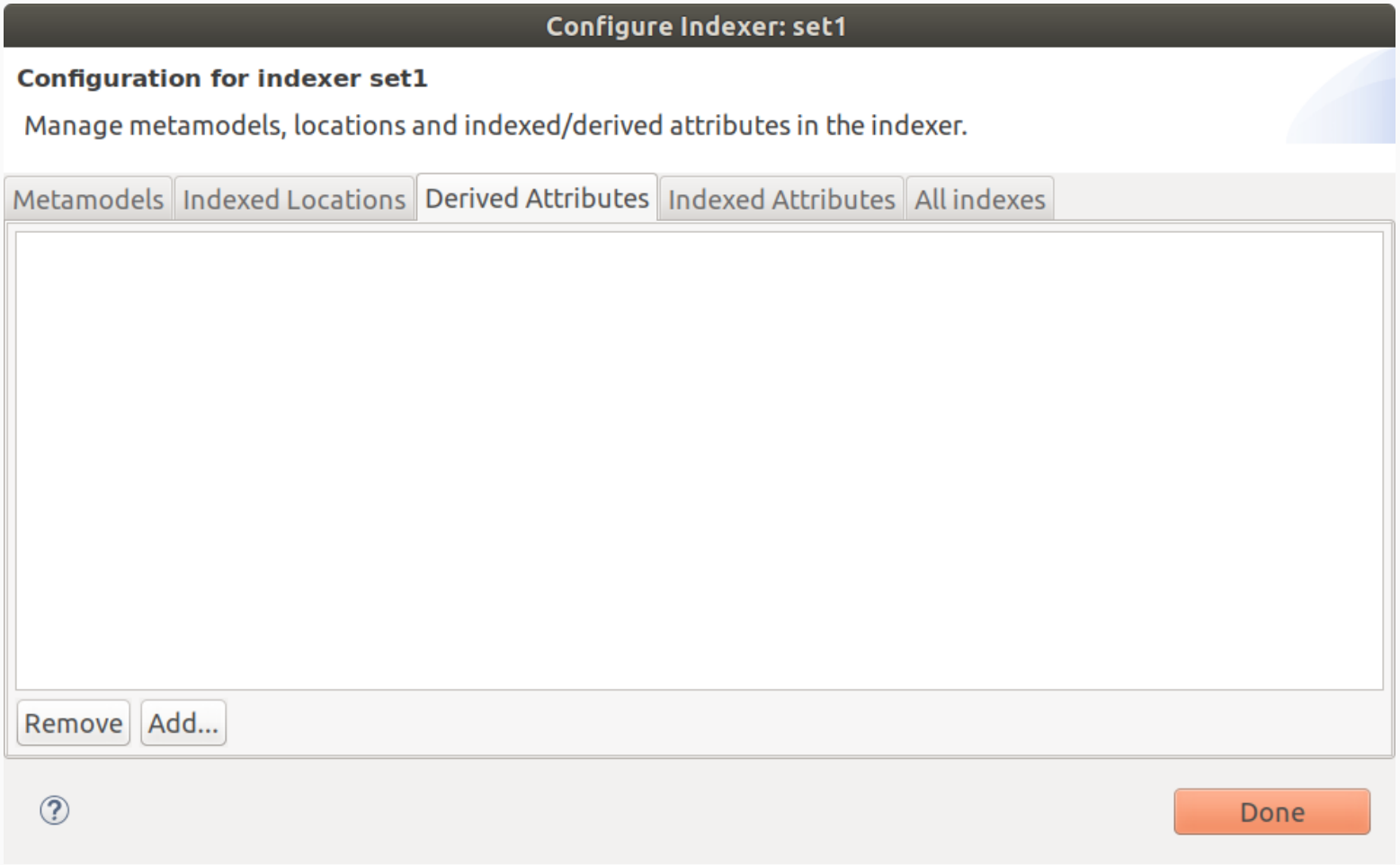
Original query (`singletons.eol`)

```
return TypeDeclaration.all.select(td | td.bodyDeclarations.exists(
  md:MethodDeclaration | md.returnType.isTypeOf(SimpleType)
    and md.returnType.name.fullyQualifiedName
      = td.name.fullyQualifiedName
    and md.modifiers.exists(mod:Modifier | mod.public = true)
    and md.modifiers.exists(mod:Modifier | mod.static = true)
));
```

Changed for MethodDeclaration derived attributes (`singletons-dmethods.eol`)

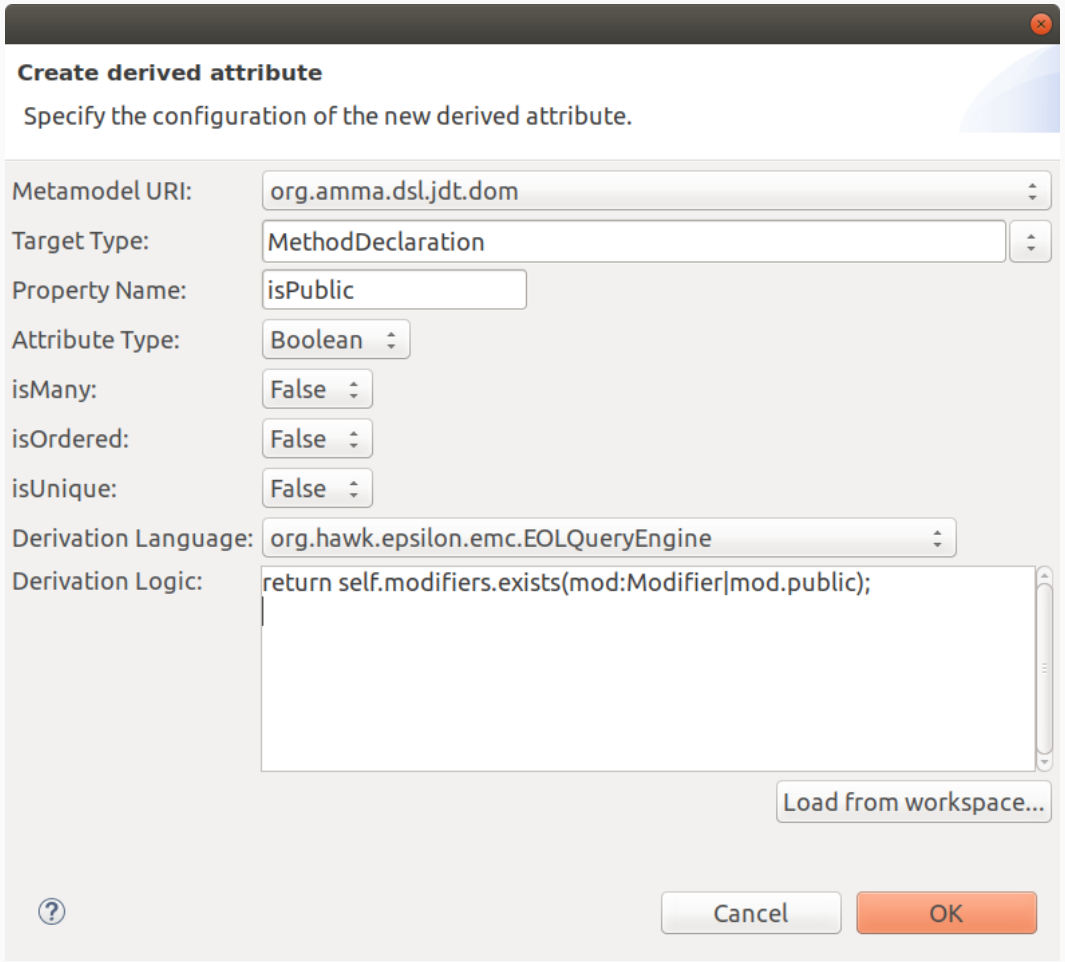
```
return TypeDeclaration.all.select(td|
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Derived attributes tab



Now we move to the "Derived Attributes" tab of the configure dialog.

Options for one of the derived attributes



For the first derived attribute, fill in the form as above. You can load the EOL code straight from the workspace — no need to copy and paste!

Derived attributes are also indexed

Revised query (`singletons-dmethods.eol`)

```
return TypeDeclaration.all.select(td|
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Can we do it faster?

- Right now, we need to go through *all* type declarations and then filter by methods
- What if we go from the methods to the types instead?
- In Hawk, top-level selects can replace iteration with lookups when using derived attributes

Use of derived attributes as index keys

Previous query (`singletons-dmethods.eol`)

```
return TypeDeclaration.all.select(td|
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Revised to use index, with derived attrs. at top level (`singletons-dmethodsindexed.eol`)

```
return MethodDeclaration.all.select(md |
  md.isPublic = true and md.isStatic = true
  and md.isSameReturnType = true
).collect( td | td.eContainer ).asSet;
```

Flagging singletons directly

Previous query (`singletons-dmethodsindexed.eol`)

```
return MethodDeclaration.all.select(md |
  md.isPublic = true and md.isStatic = true
  and md.isSameReturnType = true
).collect( td | td.eContainer ).asSet;
```

Can we do it faster?

- We could just flag types that are singletons
- This derived attribute might be less reusable, however

Final query for finding singletons

Previous query (`singletons-dmethodsindexed.eol`)

```
return MethodDeclaration.all.select(md |
   md.isPublic = true and md.isStatic = true
   and md.isSameReturnType = true
).collect( td | td.eContainer ).asSet;
```

Final query (`singletons-dtypes.eol`)

```
return TypeDeclaration.all.select(td | td.isSingleton = true);
```

Hands-on time!

This time, we will show how to use indexed and derived attributes.

Derived edges

Toy example: Person metamodel

- Person metamodel, with "parents" references.
- We want to be able to quickly find siblings, grandparents, uncles/aunts, cousins, second-cousins, ancestors...
- We can precompute this in Hawk with derived edges

Derivation logic for "grandparents" (`Person_grandparents.eol`)

We need a flat list and not a list of lists, so we use "flatten":

```
return self.parents.parents.flatten;
```

Derivation logic for "siblings" (`Person_siblings.eol`)

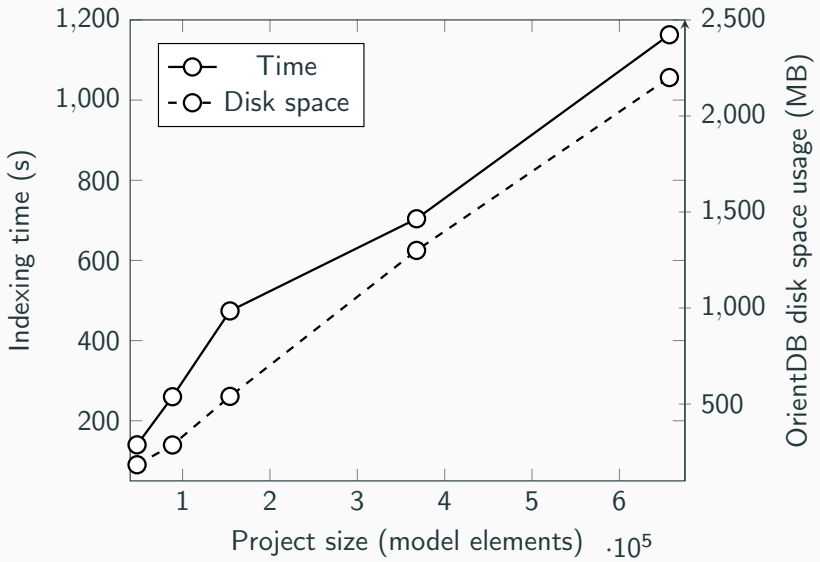
We can travel references in reverse with "revRefNav_name":

```
return self.parents.revRefNav_parents.flatten.excluding(self);
```

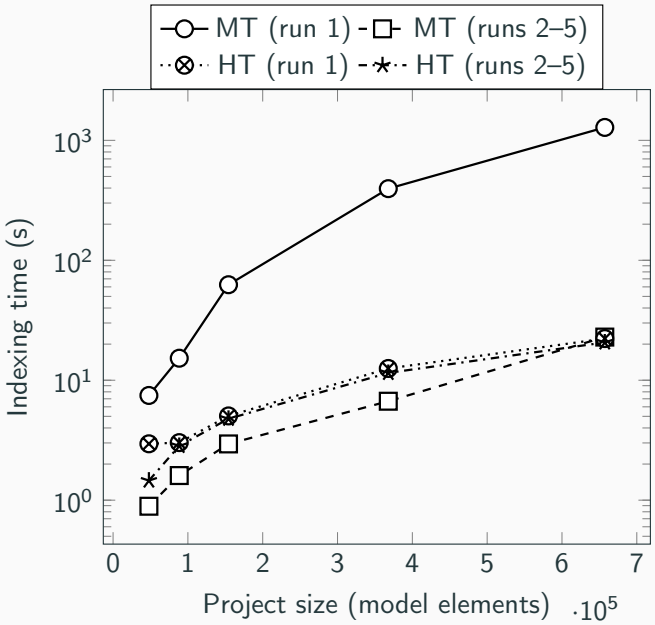
Last hands-on slot for Hawk.

We will show derived edges this time.

Hawk: integration into SOFTEAM Constellation [GDBK+16]



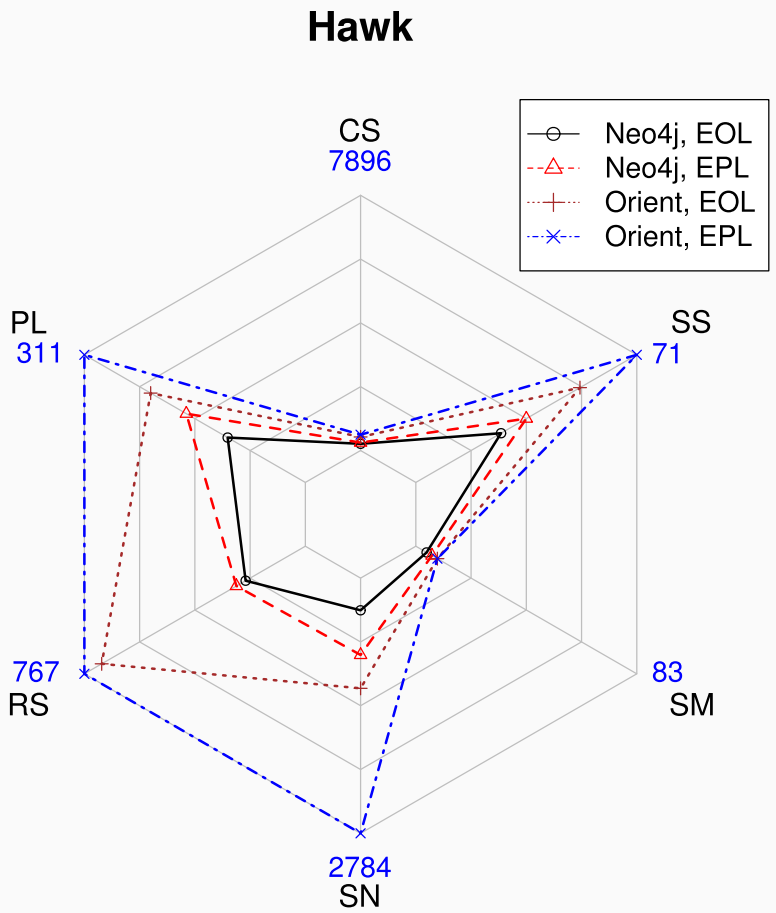
Indexing times and index sizes (OrientDB backend)

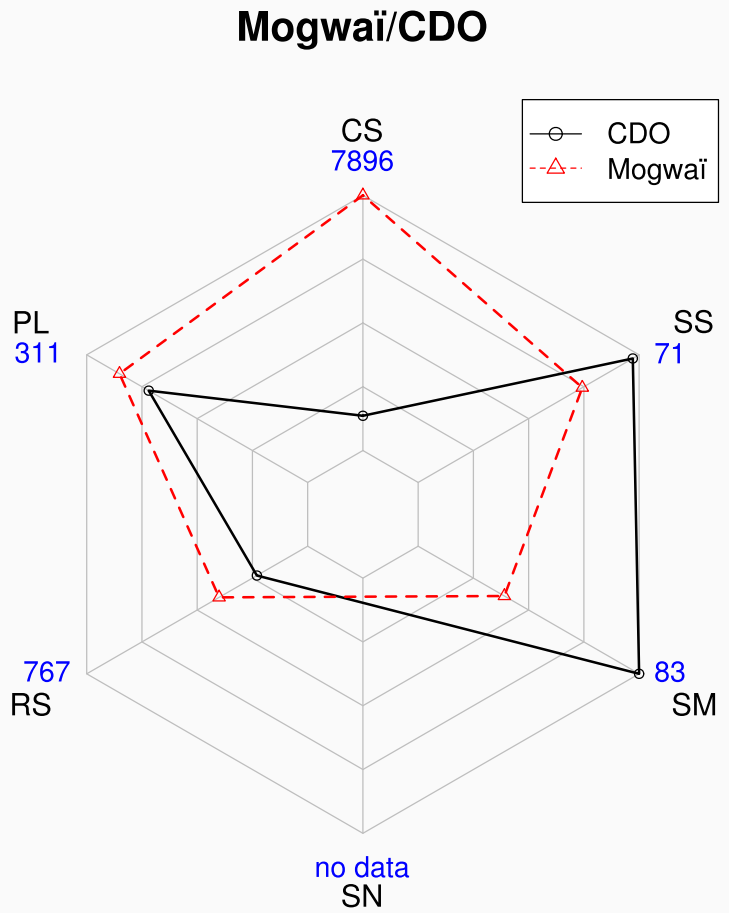


Code generation times: Modelio (MT), Hawk (HT)

- Constellation: collaboration platform over Modelio models
- SOFTEAM needed search, couldn't change persistence
- Integrated Hawk as a library: initial indexing cost quickly paid off

Stress-testing remote query APIs [GDBK+17]





- Included CDO, Hawk, Mogwaï and ranged 1–64 clients
- Reverse reference navigation was crucial in having the SN Train Benchmark [SIRV17] query run quickly

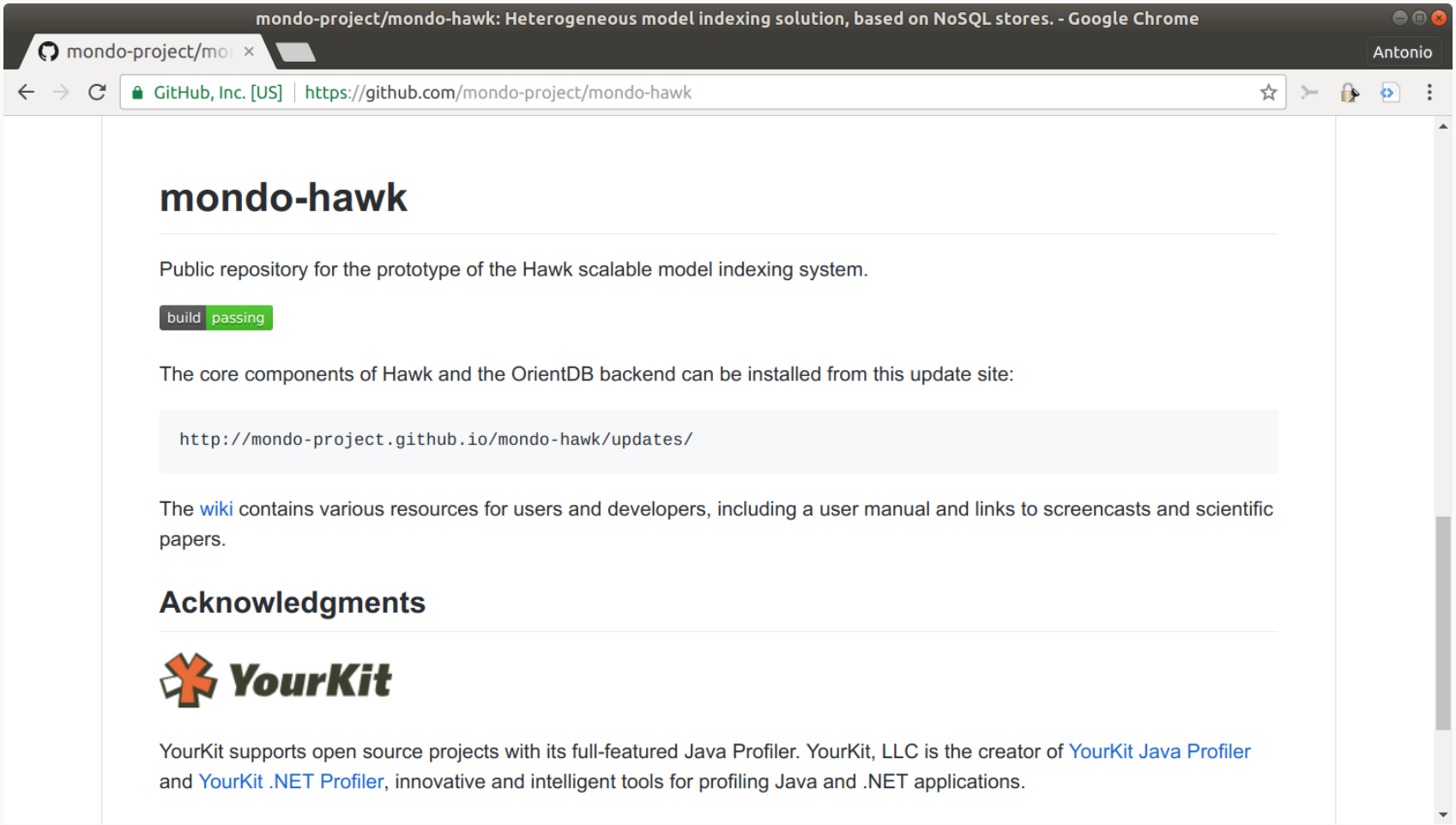
Time-aware queries in Hawk over versioned models [GDBP18]

```
var rs = RewardTableRow.latest.all.collect(row | row.getRewardShifts()).flatten();
return Sequence { rs.min(), rs.max(), rs.average() };

operation RewardTableRow getRewardShifts(): Sequence {
  var v = self.versions;
  if (v.size <= 1) { return Sequence {}; }
  else { return v.subList(0, v.size - 1).collect(v | v.value - v.prev.value); }
}
operation Sequence average() { return self.sum() / self.size(); }
```

- Extended Hawk with Greycat temporal graph support and time-aware query engine / updater components
- Can index entire Subversion-based history of a model, and ask things about its history through a new set of time-aware primitives
- Above query is finding descriptive stats for reward table shifts in a models@run.time system
- Presenting this work at 14:00 (MRT'18) — hope to see you there!

Hawk: project website



- `https://github.com/mondo-project/mondo-hawk`
- Recently accepted as Eclipse Incubator project (moving soon!)

Hawk: summing up

So far...

- Hawk is good for indexing an existing collection of model files
- You can efficiently answer queries from the index
- Indexed/derived features can be used to speed up queries

Ideas in the roadmap

- Extensible UI, covering differences in options for components
- Horizontal scaling (a flock of Hawks?)
- Web UI based on Thrift API
- More backends! (Triple stores? Neo4j 3.x? MapDB?)
- Better Git connector (JGit-based)
- Extending EOL with Linear Temporal Logic
- Visualizations based on time-aware queries

Feedback and contributions are welcome!

References i

Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S Kolovos, Marcos Aurelio Almeida da Silva, Antonin Abherve, and Alessandra Bagnato.
Integration of a graph-based model indexer in commercial modelling tools.
In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 340–350, Saint Malo, France, 2016. ACM Press.

Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S. Kolovos, Ran Wei, and Richard F. Paige.
Stress-testing remote model querying APIs for relational and graph-based stores.
Software & Systems Modeling, pages 1–29, June 2017.

References ii

A. García-Domínguez, N. Bencomo, and Luis H. García Paucar. **Reflecting on the past and the present with temporal graph-based models.** In *Proceedings of the 13th International Workshop on Models@run.time*, 2018. To be published.

Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. **The Train Benchmark: cross-technology performance evaluation of continuous model queries.** *Software & Systems Modeling*, January 2017.